Sílvia Maria Gusmão Pinto da Cunha

Algoritmo para Emparelhamento de Pares Dador-recetor em Transplantação Renal

Departamento de Matemática
Faculdade de Ciências da Universidade do Porto
outubro de 2012

Sílvia Maria Gusmão Pinto da Cunha

Algoritmo para Emparelhamento de Pares Dador-recetor em Transplantação Renal

Tese submetida à Faculdade de Ciências da Universidade do Porto para obtenção do grau de Mestre em Engenharia Matemática

Orientador: Prof. Doutor João Pedro Pedroso
Coorientadora: Prof.ª Doutora Ana Viana

Departamento de Matemática
Faculdade de Ciências da Universidade do Porto
outubro de 2012

Agradecimentos

"Determinação coragem e auto confiança são factores decisivos para o sucesso. Se estamos possuídos por uma inabalável determinação conseguiremos superá-los. Independentemente das circunstâncias, devemos ser sempre humildes, recatados e despidos de orgulho."

(Dalai Lama)

Agradeço a todos os familiares, amigos, colegas, conhecidos que direta ou indiretamente, com um pequeno ou grande gesto, nos momentos em que a coragem ou a auto confiança faltaram, me apoiaram e me mostraram o outro lado de ver as coisas. Foi assim que ultrapassei pequenas barreiras e atingi o sucesso que foi a realização desta tese.

Aos professores da instituição Faculdade de Ciências da Universidade do Porto que contribuíram para este projeto, agradeço o tempo despendido no esclarecimento de dúvidas pontuais que foram surgindo. Um agradecimento em particular para os professores João Pedro Pedroso e Ana Viana pela orientação prestada ao longo do último ano e pela experiência que me permitiram adquirir.

Por último, não podia deixar de agradecer o financiamento fornecido pela Faculdade de Ciências da Universidade do Porto para a participação na conferência COA.pt 2012.

Resumo

O Problema de Troca de Rins (Kidney Exchange Problem (KEP)) é um problema no qual pares dador/recetor incompatíveis (o recetor sofre de doença renal e necessita de um transplante) são emparelhados com outros pares incompatíveis para originar transplantes.

Quando apenas dois pares podem estar envolvidos na troca o problema pode ser resolvido em tempo polinomial usando o algoritmo de Edmond's para o problema de emparelhamento de cardinalidade máxima. O mesmo acontece quando não existe limite para o número máximo de pares (k), que podem estar envolvidos numa troca. Neste caso, o problema é resolvido utilizando uma redução ao Problema de Afetação e o Algoritmo Húngaro. Quando $k \geq 3$ e limitado, está provado que o problema é NP-completo.

Neste projeto desenvolve-se um algoritmo de Pesquisa em Árvore para resolver o KEP para o caso k=3. O problema é representado por um grafo orientado (digrafo) em que os vértices representam pares incompatíveis e os arcos representam a compatibilidade entre o dador de um par e o recetor de outro. Inicialmente os limites inferior e superior do problema são calculados através do algoritmo de Edmond's e do Algoritmo Húngaro, respetivamente. A ramificação inicia-se com a seleção de um ciclo de tamanho 3 pertencente ao digrafo inicial. É gerada uma nova solução, considerando que o ciclo selecionado faz parte da solução e os restantes (de tamanho 2) são obtidos pelo algoritmo de Edmonds. Seguidamente, em cada etapa de ramificação, é selecionado um ciclo de tamanho 3 existente no digrafo atual e o procedimento é repetido. Em todos os nós é dada preferência a ciclos de tamanho 2 devido a duas razões principais: 1) como todos os transplantes num ciclo têm de ser realizados ao mesmo tempo há várias restrições logísticas/recursos (médicos, enfermeiros, salas de operação); 2) se existem ciclos mais longos, mais pares dador/recetor são afetados se um deles desiste.

Palavras-chave: TRANSPLANTAÇÃO RENAL EMPARELHADA, ALGORITMO DE EDMONDS, PROBLEMA DE AFETAÇÃO, ALGORITMO HÚNGARO, BRANCH-AND-BOUND, PESQUISA EM PROFUNDIDADE, PESQUISA EM LARGURA, FORMULAÇÃO CICLO.

Abstract

The Kidney Exchange Problem (KEP) is a problem in which incompatible patient-donor pairs (the patient suffering from renal disease and in need of a transplant) are matched with other incompatible pairs to allow transplants.

When only two pairs are involved in an exchange the problem can be solved in polynomial time using Edmond's algorithm for maximum cardinality matching. The same happens when there is no bound on the maximum number of pairs (k) that can be involved in an exchange. In this case the problem is solved using a reduction to the assignment problem and the Hungarian Algorithm. When $k \geq 3$ and bounded, the problem is proven to be NP-complete.

In this project we develope a tree search algorithm to solve a KEP for k = 3. The problem is represented by a directed graph (digraph) in which the vertices represent incompatible pairs and the arcs represent the compatibility between a donor of one pair and a patient of another pair. Initially lower and upper bounds of the problem are computed through Edmond's algorithm and the Hungarian Algorithm, respectively. The branching is done starting by selecting one cycle of size 3 that belong to the initial digraph. For this cycle it determines the respective left and right branches, that are inserted in a waiting D. While D is not empty, it selects one of the nodes belong to D. It generates a new solution, considering that the cycle selected is part of the solution and the remaining (size 2) are obtained by the Edmond's algorithm. Then at each branching step a cycle of length 3 in the current digraph is selected and the procedure is repeated. In all nodes preference is given to cycles of size 2 due to two main reasons: 1) as all transplants in a cycle must be performed simultaneously, there are several logistic/resource constraints (doctors, nurses, operating rooms); 2) if we have longer cycles more patients/donors are affected if one pair drops out.

Keywords: KIDNEY EXCHANGE, EDMONDS ALGORITHM, ASSIGNMENT PROBLEM, HUNGARIAN ALGORITHM, BRANCH-AND-BOUND, DEPTH SEARCH, BREADTH SEARCH, CYCLE FORMULATION.

Conteúdo

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

Em Portugal existem 900.000 habitantes com doenças renais, 1 em cada 10 pessoas sofrendo de doença renal crónica. Todos os anos são registados 2.500 novos casos de Insuficiência Renal crónica terminal, existindo actualmente 16 mil doentes com a forma mais grave de Doença Renal Crónica, ou seja, a necessitar de diálise (cerca de 10 mil), ou transplantados renais (6 mil)(Associação dos Doentes Renais do Norte de Portugal, 2011).

O transplante renal é a solução que proporciona maior qualidade de vida aos portadores de insuficiência renal crónica. Contudo, nem sempre a existência de um dador disponível - mesmo que se trate de um familiar direto - significa que um transplante é viável. Fatores como a incompatibilidade entre grupos sanguíneos ou do sistema HLA (antigénios de leucócitos humanos) impedem frequentemente o transplante entre dador-recetor. Por outro lado, a espera por um dador cadáver pode tornar-se bastante longa aumentando o tempo de vida com baixa qualidade que o dador possui, ou até mesmo em alguns casos ultrapassando o tempo levando mesmo à morte do doente. A Transplantação Renal Emparelhada aumenta a possibilidade de transplantação porque permite cruzar vários pares dador/recetor, sendo cada par constituído por pessoas incompatíveis entre si, por forma a criar novos pares dador/recetor compatíveis e consequentemente permitir a realização de transplantes renais.

A portaria nº 810/2010 de 23 de Agosto criou o Programa Nacional de Doação Renal Cruzada com o objetivo de promover a dádiva de dadores vivos (portaria nº 810/2010, 2010). Neste programa, entre outros requisitos de funcionamento, entende-se por "par dador/recetor" o candidato a receber um orgão e a pessoa ou pessoas que se propõem a dar-lhe um orgão e "Doação Renal Cruzada com dador vivo" representa um processo de alocação que permite a transplantação de órgãos compatíveis através do intercâmbio de rins de dois ou mais pares dador/recetor. A regulação, normalização , controlo e fiscalização da atividade de transplantação deste programa estava até à pouco a cargo da Autoridade para os Serviços de Sangue e Transplantação (ASST), tendo recentemente sido transferida para o Instituto Português do Sangue e da Transplantação. A ASST elaborou uma circular normativa onde estão descritos todos os critérios e procedimentos que constituem este programa (ASST, 2011).

1.1 Motivação

Um ser humano saudável tem dois rins para filtrar os resíduos do sangue no corpo. Se uma pessoa tiver apenas um dos rins a trabalhar, consegue na mesma viver uma vida normal e saudável. Mas se ambos os rins deixam de trabalhar (insuficiência renal), o congestionamento de resíduos no sangue irá causar uma morte rápida. As duas soluções para a insuficiência renal são a diálise e a transplantação renal. No caso de uma pessoa ser sujeita a diálise e uma vez que esta implica passar muitas horas no hospital, a qualidade de vida dessa pessoa torna-se extremamente baixa.

Em geral, os doentes que sofrem de insuficiência renal que já estão numa fase avançada da doença recorrem à transplantação renal. Mas infelizmente, a oferta de rins de cadáveres é reduzida e o tempo de espera por um rim de cadáver é exageradamente longo. Por esta razão, mais e mais pacientes renais procuram um dador vivo (familiar ou amigo) para lhes doar um dos seus rins saudáveis. Mas, muito frequentemente, acontece que o dador vivo que um determinado paciente encontra não é compatível com ele, ou seja, o dador encontrado não pode doar o rim ao paciente em questão.

Dadas todas estas implicações e restrições que impossibilitam um maior número de transplantes renais, diversos países (Reino Unido, Estados Unidos, Holanda Portugal etc) sentiram a necessidade de fazer algo para aumentar a possibilidade de um paciente receber um rim. Desta forma, criou-se um novo plano de Transplantação Renal, nomeadamente a Transplantação Renal Emparelhada e mais concretamente em Portugal o Programa Nacional de Doação Renal Cruzada (PNDRC) no âmbito do qual surgiu este projeto.

Este problema de grande interesse prático, aliado a uma vertente de investigação forte traduz-se numa grande motivação para a elaboração deste trabalho. O problema a tratar é o caso particular da Transplantação Renal Emparelhada onde são considerados emparelhamentos de tamanho máximo 3: um problema de otimização NP-completo.

A elaboração deste algoritmo torna-se também motivante porque a formulação em Programação Inteira mais eficiente existente atualmente, Formulação Ciclo falha quando aplicada a variantes deste problema. Deste modo, pretende-se desenvolver uma abordagem diferente que se revele eficiente para as diversas variantes do problema.

1.2 Objetivo da Tese

O principal objetivo desta tese é desenvolver um algoritmo paea emparelhamento de pares dador/recetor em transplantação renal emparelhada, considerando que o tamanho máximo de um emparelhamento é 3.

O trabalho, realizado no âmbito do projeto "KEP - New models for enhancing the kidney transplantation process" financiado pela Fundação para Ciência e Tecnologia (FCT) consiste no desenvolvimento de um software avançado que permite determinar o maior número de pares dador-receptor compatíveis em programas de doação renal cruzada. As entidades que colaboram neste projeto são o INESC Porto, a Faculdade de Medicina da Universidade do Porto, a Autoridade para os Serviços de Sangue e da Transplantação, a Universidade do Minho e a Universidade de Lisboa. É no âmbito deste projeto que surge o trabalho de tese que será apresentado.

Sabe-se que, no caso da Transplantação Renal Emparelhada em que apenas são considerados emparelhamentos de tamanho dois, o problema de otimização que representa a maximização do número de transplantes realizados pode ser resolvido em tempo polinomial através do Algoritmo de Edmonds. Por outro lado, o caso em que não existe limite para o número de pares dador/recetor envolvidos num emparelhamento também pode ser resolvido em tempo polinomial utilizando o Algoritmo Húngaro.

O problema de otimização torna-se NP-completo nos casos em que o número de pares dador/recetor permitidos numa troca (k) é limitado e maior ou igual a três. Existe atualmente uma formulação em Programação Inteira (formulação ciclo) que apresenta muito bons resultados. No entanto, testes preliminares mostraram que esta abordagem não é adequada quando se verifica um conjunto de alterações na estrutura do problema. É por isso necessário explorar abordagens alternativas. Sendo assim, o desafio colocado para a elaboração desta tese é o de desenvolver um algoritmo que maximize o número de transplantes realizados para um determinado conjunto de pares, sendo que o número máximo de pares permitido num emparelhamento é três.

Outro objetivo relevante é o desenvolvimento de uma ferramenta de apoio à decisão, a usar pela entidade coordenadora do PNDRC. Tendo em conta que o processo de filtragem e emparelhamento de pares era feito manualmente, rapidamente se verificou que, com o aumento dos pares que adiram a este programa é necessário criar um processo automático de seleção.
Sendo assim foi necessário criar uma base de dados onde estão listados todos os pares dador/recetor que pretendem participar no PNDRC. Essa base de dados foi desenvolvida em Excel juntamente com a aplicação VBA e para cada um dos pares possui toda a informação necessária para efetuar as duas filtragens de compatibilidade (compatibilidade sanguínea e compatibilidade imunológica). É importante referir que essas filtragens foram ambas programadas em VBA. Os pares que resultam das duas filtragens formam um grafo que corresponde aos dados de entrada de um programa criado no otimizador CPLEX. Através deste programa, dado o grafo referido, encontra-se a solução do problema de maximizar o número de transplantes realizados. A ferramenta constituída pela base de dados e pelo programa em CPLEX está atualmente a ser utilizada pela entidade coordenadora do projeto.

Aliado ao principal objetivo desta tese surge um outro objetivo, nomeadamente o estudo e compreensão do Algoritmo de Edmonds. Isto porque, este algoritmo possui muitos conceitos específicos e de elevada complexidade que exigem um estudo bastante exaustivo e aprofundado.

O algoritmo desenvolvido consiste numa árvore de pesquisa onde são considerados os dois tipos de pesquisa: Pesquisa em Largura e Pesquisa em Profundidade. A pesquisa inicia-se considerando como nó raiz a solução do Algoritmo de Edmonds para o conjunto dos pares dador/recetor considerado, e tem como objetivo determinar sucessivamente quais os ciclos de tamanho três que podem melhorar a melhor solução atual. De uma forma mais simples, inicialmente determina-se a solução de Edmonds (solução constituída apenas por emparelhamentos de tamanho dois) para o grafo em questão. A pesquisa avança selecionando um ciclo de tamanho três para ser analisado, isto é, para verificar se através deste ciclo é possível encontrar uma solução melhor do que a solução atual. Este processo decorre iterativamente até que sejam analisados todos os ciclos de tamanho três ou até que se encontre a solução ótima.

Ao longo do algoritmo existem algumas condições que tornam a pesquisa mais inteligente e guardam a informação relevante de cada nó. Como por exemplo, determinar se vale a pena continuar a exploração de um dado nó, se o ramo que se vai analisar deve ser cortado e a pesquisa através daquele ramo termina ou por exemplo, verificar se a solução associada ao nó que está a ser analisado é melhor do que a melhor solução atual e no caso de ser melhor atualizar a melhor solução atual ou até mesmo se a solução em questão corresponde à solução ótima (situação na qual a pesquisa termina). Para a determinação do majorante em cada nó foi aplicado o Algoritmo Húngaro à formulação do Problema de Afetação correspondente.

Por último, a fim de testar a eficiência do algoritmo desenvolvido e retirar todas as conclusões necessárias, usam-se as duas versões do algoritmo (Pesquisa em Largura e Pesquisa em Profundidade) e a Formulação Ciclo (formulação de Programação Inteira Mista "forte"para o Problema em questão).

1.3 Estrutura da Tese

A estrutura desta tese é a seguinte: este capítulo é um capítulo introdutório constituído por 3 secções, a primeira secção diz respeito à motivação para a realização deste trabalho. Na segunda secção são descritos os principais objetivos atingir com a elaboração desta tese e por último a última secção diz respeito à estrutura da tese.

O capítulo 2 diz respeito à descrição do problema e é constituído por três secções. Na primeira secção é efetuada uma descrição geral do problema, onde são explicadas as incompatibilidades existentes entre os pares dador/recetor. Na segunda secção é explicado em que consiste a Transplantação Renal Emparelhada, apresentando o problema de otimização associado a este tipo de transplantação e algumas das suas variantes. Por último, na secção 3 apresenta-se a formulação matemática do problema em estudo. Começa-se por apresentar uma estrutura geral comum às diversas formulações matemáticas do problema, e de seguida são apresentadas duas Formulações existentes para o problema em questão.

No capítulo 3 descreve-se detalhadamente os conceitos teóricos que foram necessários estudar para a criação do algoritmo. Existem 3 secções, a primeira é constituída pela definição de pesquisa em árvore e pela descrição dos dois tipos de pesquisa (pesquisa em largura e pesquisa em profundidade). Na segunda secção, estão descritos todos os conceitos estudados acerca do Problema de Afetação. A terceira secção contém uma descrição aprofundada do algoritmo de Edmonds.

O capítulo 4 contém todo o trabalho desenvolvido ao longo da tese e é constituído por duas secções. Na primeira secção apresenta-se a descrição detalhada de uma ferramenta criada para determinar a solução do problema de otimização associado à Transplantação Renal Emparelhada para um determinado conjunto de pares dador/recetor. A segunda secção refere-se á descrição do algoritmo desenvolvido. Nesta secção começa-se por apresentar um exemplo prático de aplicação do algoritmo, seguidamente efetua-se uma descrição teórica das componentes do algoritmo e por último apresenta-se uma análise do algoritmo.

Os resultados computacionais são apresentados no capítulo 5. Uma vez que os dados utilizados neste projeto são obtidos através de um gerador, na secção 5.1 é explicado todo o procedimento incorporado no gerador para que este consiga gerar dados apropriados para o problema em questão. Na secção 5.2 apresentam-se os resultados computacionais resultantes da aplicação de cada um dos métodos referidos. A análise dos resultados obtidos apresenta-se na secção 5.3.

Por último, o capítulo 6 é constituído pelas conclusões finais referentes a todo o trabalho desenvolvido e pelas perspetivas de trabalho futuro que pode ser realizado no seguimento deste projeto.

Capítulo 2

Descrição do Problema

Neste capítulo primeiramente será apresentada uma descrição geral do problema a ser tratado ao longo deste trabalho. É explicado em que consiste a Transplantação Renal com dadores vivos, as principais incompatibilidades existentes entre um par dador/recetor e em particular como se realiza a transplantação renal emparelhada. De seguida, é apresentada a Formulação Matemática para este problema, sendo que primeiramente apresenta-se uma estrutura geral comum às duas formulações matemáticas descritas na literatura: Formulação Ciclo e Formulação Aresta.

2.1 Transplantação Renal

Os pacientes que sofrem de doença renal e que já estão em fase terminal necessitam de um transplante. Nesta situação existem duas possibilidades, o recurso a um dador cadáver ou o recurso a um dador vivo.

Dador Cadáver: Apesar de Portugal estar listado como o 2º país do mundo com maior número de transplantes, os pacientes têm de esperar muito tempo por um dador cadáver.

Dador Vivo: Este é alguém que conhecem e que quer dar um dos seus rins ao paciente em questão, juntos formam um par dador/recetor. No entanto, o dador apenas pode doar o seu rim se for compatível com o paciente.

As características mais importantes para avaliar a compatibilidade/ incompatibilidade entre o paciente e o dador são descritas na secção 2.1.1. Atualmente existem também algumas opções para trocar um rim mesmo que o paciente e o dador sejam incompatíveis entre si, como é descrito na secção 2.2.

2.1.1 Incompatibilidades

Existem três características importantes para avaliar a incompatibilidade entre um paciente e um dador (Gentry, 2008) (Roth et al., 2004).

A primeira característica é a incompatibilidade do grupo sanguíneo. Existem quatro tipos de grupos sanguíneos nomeadamente O, A, B, AB. Os pacientes com o grupo sanguíneo O são os mais problemáticos, isto porque estes apenas são compatíveis com dadores do grupo

sanguíneo O. Por outro lado, os pacientes com o grupo sanguíneo AB são os que têm mais vantagens, porque são compatíveis com todos os tipos de sangue.

Na tabela seguinte estão representadas todas as compatibilidades para cada tipo de sangue.

Tipo Sangue Recetor	**Tipo sangue Dador**			
	O	**A**	**B**	**AB**
O	Compatível	Incompatível	Incompatível	Incompatível
A	Compatível	Compatível	Incompatível	Incompatível
B	Compatível	Incompatível	Compatível	Imcompatível
AB	Compatível	Compatível	Compatível	Compatível

Tabela 2.1: Incompatibilidades do grupo sanguíneo dos pares dador/recetor

A segunda característica é a incompatibilidade do tipo de tecido ou incompatibilidade HLA (Antigéneos Leucócitos Humanos). O sistema HLA de cada pessoa é uma combinação de seis antigéneos: 2 antigéneos HLA-A, 2 antigéneos HLA-B e 2 antigéneos HLA-DR. Há um número de antigéneos comuns de cada tipo. Mesmo que os antígeneos do dador não sejam totalmente idênticos aos antigéneos do paciente, o dador pode mesmo assim doar o rim ao paciente, desde que todas as outras características combinem.

A terceira característica importante é o teste do crossmatch. Este teste é também feito tendo em conta o sistema HLA. Um paciente pode gerar/possuir anticorpos contra determinados antigéneos que não coincidam com os seus. Por outras palavras, um paciente não gera um anticorpo contra um dos seus próprios antigéneos, apenas pode gerar contra todos os outros antigéneos. Nesta situação, diz-se que o paciente é sensibilizado para os antigéneos para os quais contraiu anticorpos. Se um paciente é sensibilizado para o HLA do dador, isto é, possui anticorpos contra o HLA do dador, existe uma reação cruzada positiva entre o paciente e o dador e a transplantação não se pode realizar. No entanto, se o teste de crossmatch for negativo a transplantação prossegue.

Se um par dador/recetor tiver compatibilidade sanguínea, tipo de tecido compatível e crossmatch negativo, o dador e o recetor são compatíveis entre si e é possível realizar a transplantação renal entre este par. Se eles forem incompatíveis numa destas três características eles formam um par dador/recetor incompatível e não se realiza a tranplantação entre eles.

2.2 Transplantação Renal Emparelhada

A transplantação Renal Emparelhada pretende dar resposta a situações em que um doente renal não encontra um dador compatível na lista de dadores cadáveres nem um potencial dador vivo compatível consigo. Esta técnica de transplantação renal emparelha pares dador/recetor inscompatíveis com outros pares dador/recetor incompatíveis de forma a criar múltiplas trocas compatíveis.

A versão mais simples destas trocas designa-se por transplantação renal cruzada e inclui

apenas dois pares dador/recetor (Par 1, Par 2). Para que a troca ocorra, o recetor do par 1 tem de ser compatível com o dador do par 2 e o recetor do par 2 tem de ser compatível com o dador do par 1. Os pares podem agora efetuar a troca dos rins doados, como se pode visualizar na figura 2.1 e cada recetor recebe um rim a funcionar normalmente.

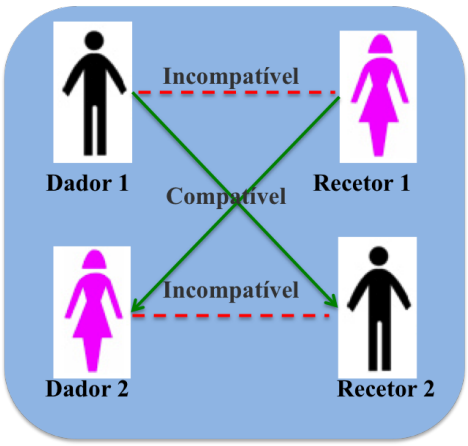


Figura 2.1: Transplantação Renal Cruzada

O conceito pode ser extendido a situações em que a troca envolve mais pares, por exemplo 3. Considere-se 3 pares para emparelhar (Par 1, Par 2, Par 3). Neste caso para além de poderem ocorrer emparelhamentos entre dois pares, podem também ser considerados emparelhamentos de tamanho 3. Para existir um emparelhamento de tamanho 3 é necessário que exista uma combinação entre os dadores e os recetores incompatíveis de forma a obterem-se três pares dador/recetor compatíveis. Por exemplo, na figura 2.2 o recetor do par 1 é compatível com o dador do par 2, o recetor do par 2 é compatível com o dador do par 3 e por último o recetor do par 3 é compatível com o dador do par 1. Assim, a troca de rins é efetuada como se pode ver na figura 2.2 e os três recetores são transplantados.

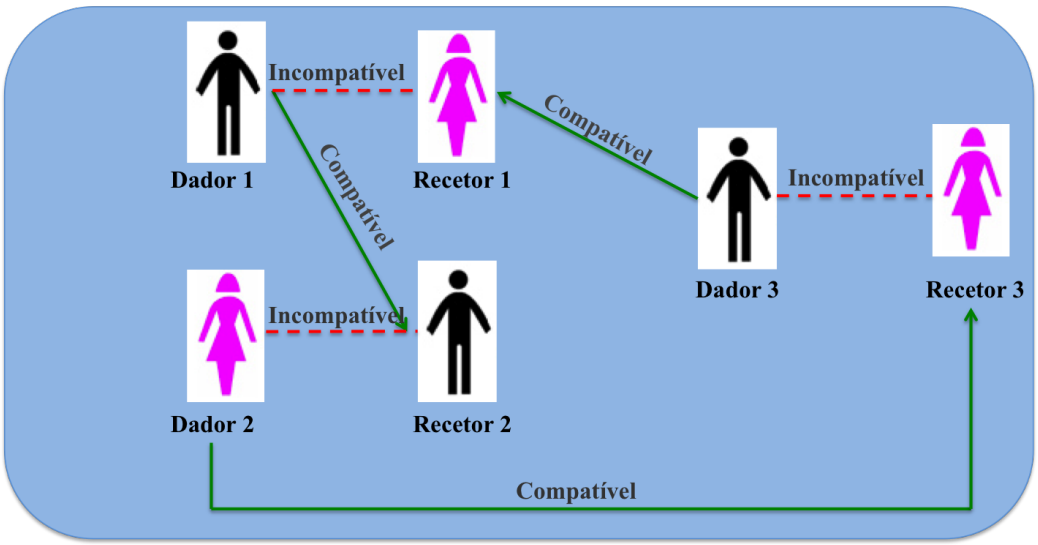


Figura 2.2: Transplantação Renal usando 3 pares

A inserção da possibilidade de haver trocas envolvendo três pares pode em determinadas situações ser bastante importante. Isto porque podem existir três pares dador/recetor (Par 1, Par 2, Par 3) incompatíveis que no caso de se considerar transplantação renal cruzada formam no máximo um conjunto de dois pares compatíveis, ou seja, realizam-se apenas dois transplantes. Mas, se forem consideradas trocas envolvendo três pares pode ser possível emparelhar o Par 1, o Par 2 e o Par 3 de forma a realizarem-se três transplantes em vez de dois.

A figura 2.3 ilustra uma situação que reforça a afirmação anterior. Considerando que o par 1 possui crossmatch positivo, os 3 pares representados na figura são incompatíveis. Se neste caso se aplicar transplantação renal cruzada tem-se o dador 1 emparelhado com o recetor 2 e o dador 2 emparelhado com o recetor 1. Ou seja, são realizados dois transplantes. Mas, ao serem consideradas trocas com três pares, obtém-se o dador 3 emparelhado com o recetor 1, o dador 1 emparelhado com o recetor 2 e o dador 2 emparelhado com o recetor 3, o que faz com que sejam realizados 3 transplantes. Verifica-se assim que a solução considerando trocas envolvendo três pares é melhor.

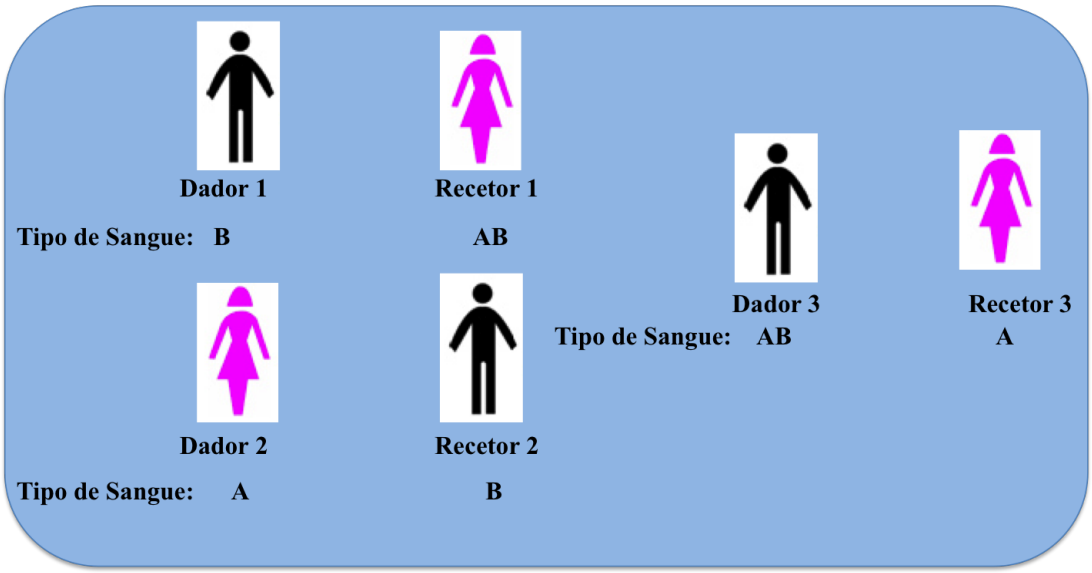


Figura 2.3: Exemplo de conjunto de pares com trocas de tamanho 3

É importante referir que um dador só irá doar o seu rim a um recetor de outro par se ele tiver a certeza que o "seu" recetor recebe o rim de um outro dador compatível. Assim, as cirurgias desses dadores (e as dos recetores) têm de ser realizadas em simultâneo. Este facto origina algumas limitações de logística/recursos, como por exemplo, a quantidade de médicos, enfermeiros, salas de operações necessários, o que faz com que seja necessário limitar o número de pares de um emparelhamento.

2.2.1 Problema de Otimização associado e suas variantes

Em programas de transplantação renal emparelhada (Kidney Exchange Program), todos os pares dador/recetor incompatíveis são listados numa base de dados. O principal objetivo é maximizar o "bem estar social" que se traduz em maximizar o número de transplantes. Em vez de se efetuar trocas usando somente dois ou três pares é possível considerar trocas entre grupos maiores de pares dador/recetor, embora não sejam aconselhadas pelas limitações referidas anteriormente. Este problema de atribuição de recetores a dadores é chamado Problema de Transplantação Renal Emparelhada.

Como referido anteriormente no objetivo da tese quando o problema envolve apenas dois pares nas trocas pode ser resolvido em tempo polinomial através do Algoritmo de Edmonds. O mesmo acontece quando não existe limite para o número máximo de pares (k) que podem ser envolvidos numa troca, utilizando uma redução ao Problema de Afetação. Quando $k \geq 3$ e limitado, está provado que o problema é NP-completo (Abraham and Blum, 2007).

Outras variantes deste problema que também estão a ser exploradas são: a possibilidade de incluir pares compatíveis, dadores altruístas (dadores que se voluntariam a doar um rim para um doente renal) e dadores cadáveres na base de dados. Ao incluir pares compatíveis há mais possibilidades para emparelhamento. Isto pode ser especialmente eficaz se um dador do grupo sanguíneo O estiver a doar a um recetor com grupo sanguíneo diferente de O no par compatível. O dador altruísta e o dador cadáver podem ser incluídos se forem vistos como um par (necessitam também de obter um rim), porque se não deixaria de existir um ciclo. O rim que é doado ao dador altruísta ou ao dador cadáver na verdade transferido para a lista de espera cadavérica ou novamente para a base de dados. Uma outra variante deste problema são as "Never Ending Altruistic Donor chains". Neste caso o dador altruísta doa o rim a um recetor de um par incompatível e o dador correspondente a esse par doa o seu rim para um outro recetor que está em lista de espera, e assim sucessivamente. A diferença desta variante relativamente às anteriores é que os transplantes não têm que ocorrer em simultâneo. Por exemplo, o dador altruísta doa o seu rim ao recetor incompatível no dia 12 de Março de 2012 e o dador desse mesmo par pode doar o rim mais tarde quando encontrar um recetor compatível (National Kidney Fundation, 2012).

2.3 Formulação Matemática

Uma abordagem promissora à resolução deste problema passa pela construção de um modelo em Programação Inteira (PI) que represente matemáticamente o problema Transplantação Renal Emparelhada. Neste trabalho serão abordadas duas das formulações em PI existentes na literatura, a Formulação Aresta e a Formulação Ciclo.

2.3.1 Estrutura Geral

Antes de formular o problema matemáticamente, é definida uma estrutura geral constituída por um grafo que reflete a compatibilidade existente entre os pares. Esta estrutura é usada

em todas as formulações matemáticas.

O problema de Transplantação Renal Emparelhada pode ser representado por um grafo bipartido em que a cada dador e a cada recetor está associado um vértice e se existe uma aresta que liga o dador i ao recetor j significa que estes são compatíveis entre si. Pode também ser representado por um grafo dirigido G=(V, A), onde V é o conjunto de todos os vértices e A o conjunto de todos os arcos do grafo. Para isso, cada par dador/recetor é codificado como sendo um vértice do grafo. E se o dador i pode doar o seu rim ao recetor j, isto é, se o dador i for compatível com o recetor j, essa compatibilidade é representada por um arco (i,j) com peso w_{ij}, onde w_{ij} representa o grau de compatibilidade entre os pares (ver figura 2.4).

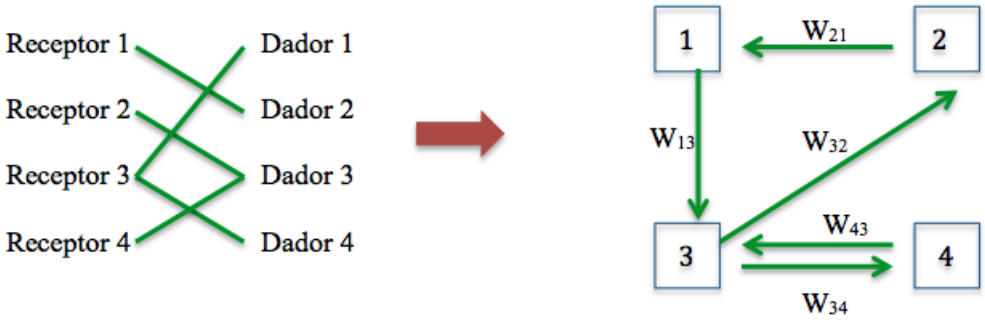


Figura 2.4: Grafo Bipartido vs Grafo Dirigido

Uma troca viável entre os pares é equivalente a um ciclo no grafo dirigido construído. O facto de existir um ciclo garante que cada par que doa um rim recebe outro em troca, caso contrário se um dador não tiver a certeza que o seu recetor incompatível vai receber um rim recusa-se a efetuar a doação. No conjunto de pares dador/recetor da figura 2.4 as trocas possíveis são 1-3-2 (em que o dador 1 doa o rim ao recetor 3, o dador 3 doa o rim ao recetor 2 e o dador 2 doa o rim ao recetor 1) e 3-4 (em que o dador 3 doa o rim ao recetor 4 e o dador 4 doa o rim ao recetor 3).

Uma troca viável no grafo pode ser constituída por mais do que um ciclo, com a restrição de que os ciclos sejam constituídos por vértices disjuntos. Por outras palavras, diferentes ciclos não podem ter vértices em comum, porque cada vértice só pode pertencer no máximo a um ciclo.

O facto de cada vértice apenas pertencer no máximo a um ciclo assegura que um dador doa no máximo um rim e um recetor também recebe no máximo um rim.

O objetivo do Problema de Transplantação Renal Emparelhada é agora encontrar uma troca de peso máximo no grafo dirigido, que consiste em encontrar o conjunto de ciclos disjuntos com tamanho máximo k, onde k é o comprimento máximo permitido para um ciclo.

Existem duas razões principais para que a restrição do tamanho máximo permitido para um ciclo surja naturalmente. A primeira surge porque, como referido anteriormente, todas as operações referentes a um ciclo têm de ser realizadas em simultâneo para certificar que nenhum dador desiste após o seu recetor incompatível receber o rim, uma vez que é difícil

fazer-se cumprir um contrato legal para a doação de órgãos. Sendo assim, o número de profissionais e de instalações necessários para que as operações ocorram em simultâneo origina a restrição logística e de recurso mencionada na Secção 2.2.
A segunda razão está relacionada com o teste de última hora realizado entre o dador e o recetor. Este teste pode revelar novas incompatibilidades que não tenham sido detetadas anteriormente e que originam o cancelamento da doação. Se essa doação estiver integrada numa troca constituída por um ciclo longo, muitos recetores serão afetados pela falha da troca em questão.

Usando a notação v_i para o vértice i, denotando o par dador/recetor incompatível i, e e_{ij} para o arco de v_i para v_j, denotando que o dador i pode doar o seu rim ao recetor j (o dador i e o recetor j são compatíveis), o Problema da Transplantação Renal pode ser agora formulado da seguinte forma:

"Tentar cobrir tantos arcos quantos possíveis através de ciclos disjuntos de tamanho máximo k."

Nas duas subsecções seguintes serão descritas duas maneiras de formular matemáticamente o Problema de Transplantação Renal Emparelhada. Estas duas formulações já foram previamente discutidas na literatura.

2.3.2 Formulação Aresta

Na Formulação Aresta (F. A.) do Problema de Transplantação Renal Emparelhada (Abraham and Blum, 2007), o problema é codificado como um problema de Programação Linear Inteira (ILP) com uma variável binária x_{ij} para cada arco existente no grafo G(V, A) correspondente:

$$x_{ij} = \begin{cases} 1 \text{ se o recetor j recebe um rim do dador i} \\ 0 \text{ caso contrário} \end{cases}$$

Como o objetivo geral é cobrir tantos arcos quantos possíveis, a Formulação Aresta tem como fim maximizar o número de arestas (pesadas) existentes na solução do problema. Em termos matemáticos este objetivo traduz-se na seguinte expressão:

$$maximizar \sum_{e_{ij} \in A} w_{ij} x_{ij}$$

Para garantir que cada par dador/recetor apenas doa um rim se também receber um rim e vice-versa é necessário considerar a seguinte restrição:

$$\sum_{j:e_{ji} \in A} x_{ji} = \sum_{j:e_{ij} \in A} x_{ij} \quad \forall\, i \in V$$

Esta restrição designa-se por *"Restrição de Conservação"* e garante que em cada par dador/recetor o número de rins doados pelo dador é igual ao número de rins recebidos pelo recetor.

Cada dador apenas pode doar um rim, por isso é necessário também garantir que cada dador doa no máximo um rim. Essa garantia é obtida através da *"Restrição da Capacidade"* que é apresentada de seguida:

$$\sum_{i:e_{ij}\in A} x_{ij} \leqslant 1 \quad \forall\, j \in V$$

Por último, a restrição que falta referir é a restrição do tamanho máximo dos ciclos. Para incluir apenas ciclos de tamanho inferior ou igual a k é necessário excluir todos os caminhos existentes no grafo de tamanho igual ou superior a k (mas caminhos de tamanho k-1 têm de ser considerados, uma vez que podem originar ciclos de tamanho k). A restrição que traduz esta limitação do tamanho máximo dos ciclos é a seguinte:

$$\sum_{1\leq j\leq k} x_{i_j i_{j+1}} \leqslant k-1 \quad \forall\, i_1 i_2 ... i_k i_{k+1}\ com\ i_1 \neq i_{k+1}$$

Esta última restrição é a mais difícil porque implicitamente requer todos os k-caminhos e consequentemente implica a existência de um algoritmo para encontrar esses caminhos. Como o número de k-caminhos pode ser muito grande, haverá muitas restrições dessas e o tempo necessário para gerar todos os caminhos será muito elevado.

Pode-se então resumir a Formulação Aresta do Problema de Transplantação Renal Emparelhada no seguinte quadro:

$$\text{maximizar} \quad \sum_{e_{ij}\in A} w_{ij}x_{ij} \tag{2.1a}$$

$$\text{sujeito a} \quad \sum_{j:e_{ji}\in A} x_{ji} = \sum_{j:e_{ij}\in A} x_{ij} \qquad (\forall\ \text{i} \in \text{V}) \tag{2.1b}$$

$$\sum_{i:e_{ij}\in A} x_{ij} \leqslant 1 \qquad (\forall\ \text{j} \in \text{V}) \tag{2.1c}$$

$$\sum_{1\leqslant j\leqslant k} x_{i_j i_{j+1}} \leqslant k-1 \qquad (\forall\ i_1\ i_2\ ...\ i_k\ i_{k+1}\ \text{com}\ i_1 \neq i_{k+1}) \tag{2.1d}$$

$$x_{ij} \in \{0,1\} \qquad (\ \forall\ e_{ij} \in \text{A}) \tag{2.1e}$$

Para facilitar a compreensão da Formulação Aresta segue-se um exemplo concreto com a formulação completa. Considere-se o grafo representado na Figura 2.5.

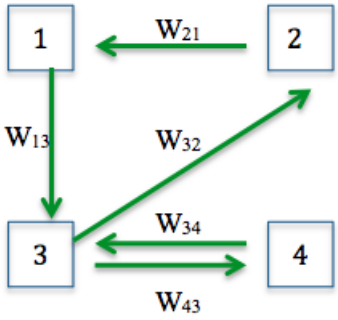


Figura 2.5: Exemplo Formulação Aresta

Para este caso se se considerar que $w_{ij} = 1 \, \forall \, e_{ij} \in A$ e que não existe limite para o tamanho dos ciclos admissíveis, a Formulação Aresta é a seguinte:

$$\begin{aligned}
&\text{maximizar} && \mathbf{z} = x_{21} + x_{32} + x_{13} + x_{34} + x_{43} && \text{(2.2a)}\\
&\text{sujeito a} && x_{21} = x_{13} \quad \text{(nó 1)} && \text{(2.2b)}\\
& && x_{32} = x_{21} \quad \text{(nó 2)} && \text{(2.2c)}\\
& && x_{13} + x_{43} = x_{32} + x_{34} \quad \text{(nó 3)} && \text{(2.2d)}\\
& && x_{43} = x_{34} \quad \text{(nó 4)} && \text{(2.2e)}\\
& && x_{13} \leq 1 \quad \text{(nó 1)} && \text{(2.2f)}\\
& && x_{21} \leq 1 \quad \text{(nó 2)} && \text{(2.2g)}\\
& && x_{32} + x_{34} \leq 1 \quad \text{(nó 3)} && \text{(2.2h)}\\
& && x_{43} \leq 1 \quad \text{(nó 4)} && \text{(2.2i)}
\end{aligned}$$

Ao analisar o quadro anterior verifica-se que as quatro primeiras restrições referem-se ao facto de que em cada par o dador apenas doa um rim se o seu recetor recebe outro em troca e que as quatro restrições seguintes representam a limitação de que cada dador apenas pode doar no máximo um rim.

2.3.3 Formulação Ciclo

Na Formulação Ciclo (F. C.) do Problema de Transplantação Renal Emparelhada (Abraham and Blum, 2007), o problema é codificado como um problema de ILP com uma variável binária x_c por cada ciclo de comprimento inferior ou igual a k existente no grafo G(V, A) correspondente:

$$x_c = \begin{cases} 1 \text{ se o ciclo c é selecionado} \\ 0 \text{ caso contrário} \end{cases}$$

Uma vez que o objetivo geral é cobrir tantos arcos quantos possíveis, o objetivo da formulação ciclo não pode depender apenas do número de ciclos incluídos na solução, mas também do

comprimento desses ciclos. Assim, multiplicando a variável de decisão x_c pelo comprimento do ciclo c o objetivo torna-se indiretamente novamente na maximização do número de arestas existentes na solução, e representa-se pela seguinte expressão:

$$maximizar \sum_{c \in C(k)} w_c x_c, \quad \text{onde } w_c \text{ representa o comprimento do ciclo c}$$

onde $C(k)$ representa o conjunto de todos os ciclos com tamanho máximo k.

O único conjunto de restrições que é necessário considerar é que cada par dador/recetor pertence apenas no máximo a um ciclo, isto é:

$$\sum_{c: v_i \in c} x_c \leqslant 1 \quad \forall\, v_i \in V$$

Deste modo, em cada par dador/recetor o dador só vai doar no máximo um rim (e o recetor recebe no máximo um rim). Pelo facto de se usarem só ciclos a restrição (2.1b) é automaticamente satisfeita. E por último, ao incluirem-se apenas ciclos de comprimento máximo k a restrição (2.1d) também é satisfeita. Posteriormente será dado um exemplo concreto da Formulação Ciclo, onde também será explicada a verificação das restrições mencionadas.

A dificuldade desta formulação é a necessidade explícita de ter todos os ciclos antes de se iniciar a otimização, uma vez que os ciclos são as variáveis de decisão. Sendo assim, a adição de um vértice extra no problema pode causar um crescimento elevado do número de ciclos, o que fará com que haja um aumento considerável no número de variáveis e seja mais difícil encontrar todos os ciclos.

Posto isto e considerando que w_c representa o comprimento do ciclo c, a Formulação Ciclo resume-se ao seguinte:

$$\text{maximizar} \quad \sum_{c \in C(k)} w_c x_c \tag{2.3a}$$

$$\text{sujeito a} \quad \sum_{c: v_i \in c} x_c \leqslant 1 \qquad (\forall\, v_i \in \text{V}) \tag{2.3b}$$

$$x_c \in \{0, 1\} \qquad (\ \forall\ \text{c} \in \text{C(k)}) \tag{2.3c}$$

Para facilitar a compreensão da formulação ciclo segue-se um exemplo concreto com a formulação completa. Na Figura 2.6 estão representados dois grafos diferentes que traduzem o mesmo problema. O grafo da direita apresenta cada um dos ciclos admissíveis existentes no exemplo através da notação C_i.

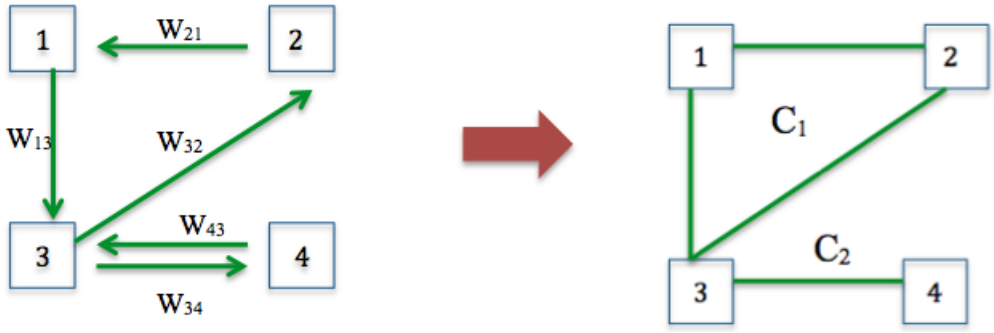


Figura 2.6: Exemplo Formulação Ciclo

Para este caso a formulação ciclo é a seguinte:

$$\begin{aligned}
&\text{maximizar} && \mathbf{z} = 3x_{c_1} + 2x_{c_2} && \text{(2.4a)}\\
&\text{sujeito a} && x_{c_1} \leq 1(\text{nó } 1) && \text{(2.4b)}\\
& && x_{c_1} \leq 1(\text{nó } 2) && \text{(2.4c)}\\
& && x_{c_1} + x_{c_2} \leq 1(\text{nó } 3) && \text{(2.4d)}\\
& && x_{c_2} \leq 1(\text{nó } 4) && \text{(2.4e)}
\end{aligned}$$

Pode-se constatar que existe uma restrição para cada nó do grafo que garante que cada par dador/recetor pertence no máximo a um ciclo (o nó 3 pode pertencer ao ciclo 1 ou ao ciclo 2 mas nunca aos dois em simultâneo). Falta então verificar as restrições de Conservação e de Capacidade. Para isso apresenta-se a Figura 2.7 onde está representado o grafo referente ao exemplo em questão. Sendo que as arestas que pertencem ao ciclo C_1 estão desenhadas a azul e as arestas que pertencem ao ciclo C_2 a cor verde. O grafo foi desenhado desta forma para que seja mais fácil perceber a ligação entre as variáveis de cada uma das formulações.

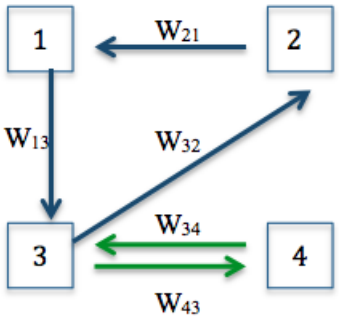


Figura 2.7: Exemplo Formulação Ciclo

Para a "Restrição de Conservação"supõe-se que, por exemplo, $x_{c_1} = 1$ o que implica que $x_{21} = x_{32} = x_{13} = 1$, logo a "Restrição de Conservação"é verificada. Para o caso de $x_{c_2} = 1$ o raciocínio é análogo, obtendo-se o seguinte esquema:

$$x_{c_1} = 1 \Rightarrow x_{21} = x_{32} = x_{13} = 1$$

$$x_{c_2} = 1 \Rightarrow x_{34} = x_{43} = 1$$

No caso da "Restrição de Capacidade"considerando-se por exemplo o vértice 3, sabe-se que este vértice apenas pode pertencer a um dos ciclos existentes e portanto ou é selecionada a aresta x_{32} ou é selecionada a aresta x_{34} o que garante que o dador 3 apenas doa um rim. Tal raciocínio está representado no seguinte esquema:

$$\underbrace{x_{c_1} + x_{c_2} \leq 1}_{\textbf{restrição F.C.}} \Rightarrow \underbrace{x_{32} + x_{34} \leq 1}_{\textbf{restrição F.A.}}$$

Capítulo 3

Noções Teóricas Estudadas

Este capítulo consiste na descrição detalhada dos conceitos e dos métodos teóricos que foram necessários estudar para o desenvolvimento do algoritmo apresentado no **Capítulo 4**. Estes conceitos foram: o Algoritmo pesquisa em árvore (e os dois tipos básicos de pesquisa em árvore, pesquisa em largura e pesquisa em profundidade), o Algoritmo de Edmonds e o Problema de Afetação (abordando também a unimodularidade da matriz das restrições associada a este problema). Nas três próximas secções serão descritos cada um dos métodos e conceitos teóricos referidos.

3.1 Algoritmo Pesquisa em Árvore

O algoritmo de pesquisa em árvore pode ser utilizado para encontrar soluções ótimas para vários problemas de otimização, especialmente em otimização combinatória. Este algoritmo consiste numa enumeração sistemática de todos os candidatos a solução, através da qual candidatos a solução infrutíferos são descartados tendo em conta os limites superior ou inferior da quantidade que está a ser otimizada.

Descrição Geral

Para facilitar a descrição do algoritmo considere-se um caso concreto onde o objetivo do problema é minimizar a função $f(x)$, x pertence a um conjunto de soluções admissíveis ou candidatas S (ao qual se dá o nome de região admissível ou espaço de pesquisa).
Um procedimento de pesquisa em árvore requer duas ferramentas. A primeira é o procedimento de divisão que, dado o conjunto das soluções candidatas S, retorna dois ou mais subconjuntos $S_1, S_2, ..., S_n$, cuja união destes é S. Note-se que o mínimo de $f(x)$ em S é o mínimo de $\{v_1, v_2, ..., v_n\}$, onde cada v_i é o mínimo de $f(x)$ no subconjunto S_i. Esta etapa é designada por ramificação ("branching"). Através da aplicação recursiva da ramificação, define-se uma estrutura de árvore (árvore de pesquisa), cujos nós são os subconjuntos de S. A segunda ferramenta é o procedimento que calcula o limite inferior e o limite superior para o valor mínimo de $f(x)$ dentro de um dado subconjunto de S. A designação atribuída a esta etapa é "limitadora"("bounding").

A ideia fundamental do algoritmo de pesquisa em árvore é a seguinte: se o limite inferior para

o conjunto de soluções candidatas de algum nó da árvore A é maior do que o limite superior para algum outro nó B, então A pode ser seguramente excluído. A esta fase chama-se poda ("pruning") e geralmente é implementada através do registo do limite superior mínimo (m) obtido entre todos os nós analisados até aquele instante; qualquer nó cujo limite superior seja maior do que o valor m pode ser descartado.

O algoritmo para quando o conjunto de soluções candidatas S é reduzido a um único elemento, ou quando o limite superior e o limite inferior do conjunto S coincidem.

Em cada nível do algoritmo é necessário decidir qual o próximo nó a ser analisado detalhadamente. Para tomar essa decisão existem duas opções básicas: pesquisa em largura e pesquisa em profundidade. Cada uma destas opções será descrita de seguida.

Pesquisa em largura

A pesquisa em largura é uma variante da pesquisa em árvore em que se começa pelo nó raiz e se exploram todos os seus descendentes. Então, para cada um desses descendentes, exploram-se os seus descendentes inexplorados, e assim por diante até que se encontre o alvo da pesquisa.

A figura 3.1 ilustra como é percorrida uma árvore considerando pesquisa em largura. Cada nó foi numerado de acordo com a ordem por que foi visitado. Um exemplo de aplicação deste tipo de pesquisa pode ser visualizado no **Apêndice A.1**.

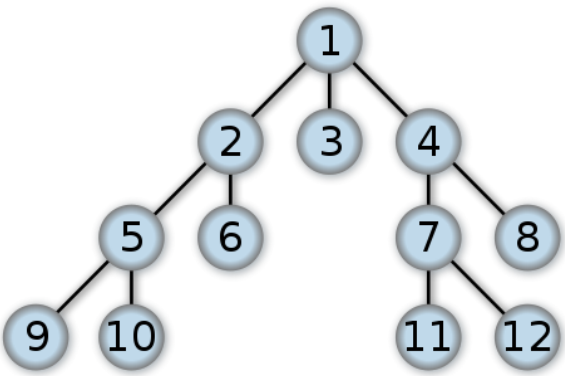


Figura 3.1: Árvore com Pesquisa em Largura

Pesquisa em Profundidade

O algoritmo de pesquisa em profundidade progride através da expansão do primeiro nó filho da raiz da árvore de pesquisa, que se aprofunda cada vez mais, até que se encontre o alvo da pesquisa ou até se deparar com um nó que não possui filhos (nó folha). Nesta situação retrocede-se no caminho deste nó em direção à raiz até se encontrar o primeiro nó filho (caso exista) que ainda não tenha sido explorado.

A Figura 3.2 ilustra o esquema de uma pesquisa em profundidade, onde cada nó foi numerado de acordo com a ordem por que foi visitado. Para este tipo de pesquisa no **Apêndice A.2** encontra-se um exemplo de aplicação.

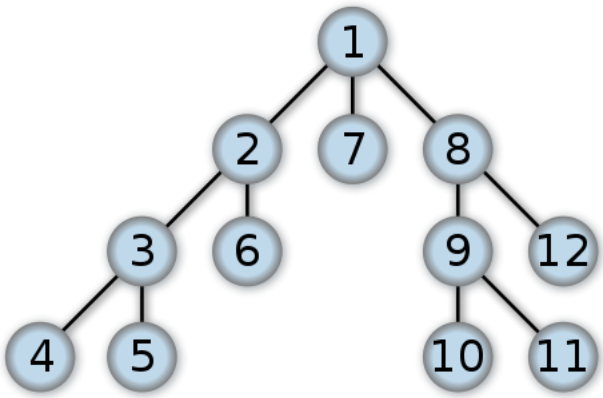


Figura 3.2: Esquema Pesquisa em Profundidade

3.2 Problema de Afetação

3.2.1 Descrição

O problema de afetação linear (linear assignment problem) pode-se descrever da seguinte forma: dada uma matriz de custos C $n \times n$, o objetivo é emparelhar cada uma das linhas com uma coluna diferente (duas linhas não podem ter a mesma coluna), de tal maneira que o custo desse emparelhamento seja minimizado. Pretende-se assim, selecionar n elementos da matriz C garantindo que existe no máximo um elemento selecionado em cada linha e em cada coluna e a soma dos custos correspondentes aos elementos selecionados seja mínima.

Alternativamente o problema de afetação linear pode ser definido através de um modelo de teoria de grafos. Define-se um grafo bipartido $G(U, V; E)$, onde a cada linha da matriz C está associado um vértice em U, a cada coluna da matriz C está associado um vértice em V e a entrada c_{ij} da matriz representa o custo da aresta (i,j), isto é, o custo de associar o vértice i ao vértice j, onde $i, j = 1, ..., n$ e $i \in U$ e $j \in V$. O problema torna-se então em determinar o emparelhamento perfeito de custo mínimo em G. Designa-se por emparelhamento perfeito

um emparelhamento que possua cardinalidade $\lfloor \frac{|V|}{2} \rfloor$. Por outras palavras, torna-se no problema de emparelhamento pesado em grafos bipartidos que consiste em encontrar um subconjunto de arestas, de tal forma que cada vértice pertence exatamente a uma aresta e a soma dos custos (pesos) destas arestas é a menor possível.

Sem perda de generalidade, normalmente assume-se que os custos c_{ij} são todos não negativos. Os casos em que existem custos negativos podem ser tratados através da adição do valor $x = -min\{c_{ij}\}$ a cada elemento de C: uma vez que é necessário selecionar um elemento em cada linha, qualquer solução de valor z para a matriz de custos original corresponde a uma solução de valor $z + nx$ para a matriz de custos transformada. Desta forma, pode-se resolver a versão deste problema cujo objetivo é de maximização, efetuando a transformação $c_{ij} = -c_{ij}$ para cada um dos elementos de C (esta propriedade será utilizada na secção 3.2.3).

Em geral, assume-se também que todos os valores de C são finitos, sendo que em determinadas situações como por exemplo, quando existe uma aresta proibida (i,j), ou então quando a matriz de custos se trata de uma matriz nxn (onde n é o número de vértices) e não existe uma determinada aresta (i,j) no grafo, torna-se necessário atribuir um custo c_{ij} muito elevado a essa aresta para que assim esta nunca seja selecionada (esta propriedade será utilizada na secção 3.2.3). Concluída a descrição do problema segue-se a formulação matemática deste.

Formulação Matemática

Para formular matemáticamente este problema é necessário definir uma variável binária x_{ij} tal que:

$$x_{ij} = \begin{cases} 1 \text{ se a linha i é afetada à coluna j} \\ 0 \text{ caso contrário} \end{cases}$$

Equivalentemente, $x_{ij} = 1$ significa que a aresta (i,j) do grafo bipartido é selecionada, e $x_{ij} = 0$ significa que a aresta não foi selecionada.

Sendo assim, possuindo a matriz de custos C e sabendo que o objetivo do problema é minimizar os custos de afetação, a função objetivo é representada pela seguinte expressão:

$$minimizar \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$

A condição de que cada linha é afetada exatamente a uma coluna, e vice-versa, em termos do grafo bipartido traduz-se em que a cada vértice do conjunto U corresponde exatamente um vértice do conjunto V e vice-versa. Para garantir tal condição é necessário considerar as duas seguintes restrições:

$$\sum_{j=1}^{n} x_{ij} = 1 \text{ i= 1,2,...,n}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \quad \text{j= 1,2,...,n}$$

Posto isto, a formulação matemática do problema de afetação linear apresenta-se resumida no seguinte quadro:

$$\text{minimizar} \quad \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \tag{3.1a}$$

$$\text{sujeito a} \quad \sum_{j=1}^{n} x_{ij} = 1 \qquad \text{(i= 1,2,...,n)} \tag{3.1b}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad \text{(j= 1,2,...,n)} \tag{3.1c}$$

$$x_{ij} \in \{0,1\} \qquad (i,j = 1,2,...,n) \tag{3.1d}$$

Para clarificar toda a descrição efetuada sobre o problema de afetação linear segue-se um problema concreto e a sua respetiva formulação matemática.

Suponha-se que existem 3 empregados e 3 empregos, onde a cada emprego tem de ser atribuído exatamente um empregado; e a cada empregado também tem de ser atribuído um emprego. O custo de atribuir o emprego i ao empregado j é dado pela entrada c_{ij} da seguinte matriz C:

$$C = \begin{bmatrix} 3 & 4 & 2 \\ 4 & 2 & 5 \\ 9 & 7 & 15 \end{bmatrix}$$

Como se pode verificar através da análise da matriz C, todos os empregados têm competências para trabalhar em qualquer um dos empregos disponíveis, por isso o grafo bipartido que representa este problema é o $K_{3,3}$ (grafo bipartido completo que contém duas partições de 3 vértices) representado na imagem 3.3.

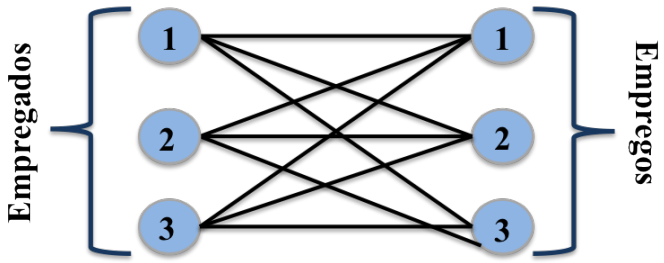


Figura 3.3: Grafo representativo do Problema de Afetação

Este problema traduz-se matemáticamente na seguinte formulação:

$$\text{minimizar} \quad \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \tag{3.2a}$$

$$\begin{aligned}
\text{sujeito a} \quad & x_{11} + x_{12} + x_{23} = 1 \quad \text{(emprego 1)} && \text{(3.2b)}\\
& x_{21} + x_{22} + x_{23} = 1 \quad \text{(emprego 2)} && \text{(3.2c)}\\
& x_{31} + x_{32} + x_{33} = 1 \quad \text{(emprego 3)} && \text{(3.2d)}\\
& x_{11} + x_{21} + x_{31} = 1 \quad \text{(empregado 1)} && \text{(3.2e)}\\
& x_{12} + x_{22} + x_{32} = 1 \quad \text{(empregado 2)} && \text{(3.2f)}\\
& x_{13} + x_{23} + x_{33} = 1 \quad \text{(empregado 3)} && \text{(3.2g)}
\end{aligned}$$

Para finalizar, é necessário explicar como o problema da Transplantação Renal Emparelhada pode ser resolvido através do problema de afetação. Na secção 3.2.3, serão apresentadas detalhadamente todas as transformações que são necessárias efetuar na função objetivo e na matriz de custos do Problema de Afetação para que através deste se possa determinar a solução do problema em estudo. Nesta secção explica-se como o problema do emparelhamento num grafo não bipartido $G(V, E)$ pode ser representado por um problema de emparelhamento num grafo bipartido.

No grafo G representado no lado esquerdo da Figura 3.4, cada vértice representa um par dador/recetor, isto é, o vértice 1 representa o $Par_1 = (Dador_1, Recetor_1)$. Este grafo pode ser transformado num grafo bipartido $H = (U_1, V_1, E_1)$, onde $|U_1| = |V_1| = |V|$, apresentado no lado direito da Figura 3.4. No conjunto U_1 estão representados os dadores, no conjunto V_1 estão representados os recetores, e por cada arco (i,j) existente em G, existe uma aresta que liga o vértice $j \in U_1$ ao vértice $i \in V_1$. Toda esta transformação pode ser visualizada na seguinte imagem:

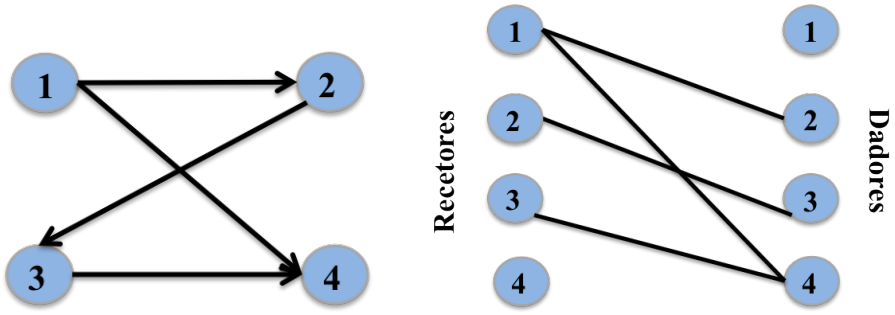


Figura 3.4: Grafo não Bipartido Problema Emparelhamento e o respetivo Grafo Bipartido

3.2.2 Unimodularidade e o Problema de Afetação

Para a elaboração desta subsecção as referências utilizadas foram as seguintes: (Garfinkel and Nemhauser, 1972) (Nemhauser and Wolsey, 1999).

De uma maneira geral, não é fácil obter soluções ótimas para os problemas lineares que correspondem a formulações de problemas de optimização combinatória porque se impõe adicionalmente a integralidade da solução ótima procurada. Tais problemas são geralmente designados por problemas de optimização linear inteira.

Se a restrição de integralidade for ignorada, então pode-se resolver o problema linear resultante, denotado por Relaxação Linear, através de, por exemplo, o método Simplex. Este método iterativo desloca-se de ponto extremo em ponto extremo até alcançar a solução ótima que, garantidamente, é um ponto extremo. Se for possível garantir que todos os pontos extremos são definidos por números inteiros então também a solução ótima será inteira. Consequentemente ao resolver-se a Relaxação Linear, resolve-se também o problema original. Uma condição suficiente para que isso aconteça é a matriz das restrições ser Totalmente Unimodular.

Posto isto, primeiramente será apresentada a definição de Matriz Totalmente Unimodular e um dos teoremas através do qual se prova que uma dada matriz é Totalmente Unimodular. Seguidamente, e utilizando a informação referida anteriormente, irá ser provada a Unimodularidade Total da matriz das restrições do Problema de Afetação e através da demonstração de um teorema, serão deduzidas as vantagens obtidas pelo facto da matriz em questão ser totalmente unimodular.

Matriz Totalmente Unimodular

Definição: *Uma matriz diz-se Totalmente Unimodular se uma qualquer submatriz quadrada tiver determinante 0, ± 1.*

Todos os elementos de uma matriz Totalmente Unimodular são zeros, uns e menos uns.
As matrizes A_1 e A_2 abaixo são Totalmente Unimodulares enquanto que as matrizes A_3 e A_4 não são, porque no caso da matriz A_3 o $det(A_3)$=2. Para a matriz A_4 a condição necessária para que esta matriz seja totalmente unimodular falha porque para a submatriz quadrada $A_{41} = \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}$ o determinante possui valor 2.

$$A_1 = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}, \quad A_2 = \begin{bmatrix} 1 & -1 & 1 \\ 0 & 1 & -1 \end{bmatrix}$$

$$A_3 = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & -1 \\ 1 & 0 & 1 \end{bmatrix}, \quad A_4 = \begin{bmatrix} 1 & 0 & 1 \\ -1 & 1 & 1 \end{bmatrix}$$

Existem diversos teoremas através dos quais é possível provar que uma dada matriz é Totalmente Unimodular, no entanto nesta secção apenas será apresentado e demonstrado aquele que será usado para provar a Unimodularidade Total da matriz de restrições do Problema de Afetação.

Teorema 3.2.2.1: *Seja A uma matriz inteira cujas entradas pertencem ao conjunto { 0,1,-1}. A matriz A é totalmente unimodular se:*

1) *Cada coluna da matriz não contém mais do que 2 elementos diferentes de 0;*

2) *As linhas da matriz podem ser particionadas em dois subconjuntos Q_1 e Q_2 tais que:*

a) *Se uma coluna contém dois elementos diferentes de 0 com o mesmo sinal, um elemento pertence a cada um dos subconjuntos;*

b) *Se uma coluna contém dois elementos diferentes de 0 com sinais opostos, ambos pertencem ao mesmo subconjunto.*

Prova: *A prova é feita por indução matemática. É claro que, qualquer submatriz de A constituída apenas por um elemento possui determinante igual a 0,1 ou -1. Assuma-se que o teorema é válido para todas as submatrizes de A de ordem menor ou igual a $k-1$. Considere-se uma submatriz arbitrária de ordem k, C. Se C contém o vetor nulo, então $det(C) = 0$. Se C contém uma coluna com apenas um elemento diferente de 0, expande-se $det(C)$ através dessa coluna e aplica-se a hipótese de indução. Finalmente, considere-se o caso no qual todas as colunas de C contêm dois elementos diferentes de 0. Note-se que a partir de* ***2a)*** e **2b)** para cada coluna j tem-se:

$$\sum_{i \in Q_1} a_{ij} = \sum_{i \in Q_2} a_{ij} \quad j = 1, ..., k \tag{3.3}$$

Seja r_i a i-ésima linha da matriz C. Por 3.3 $\sum_{i \in Q_1} r_i - \sum_{i \in Q_2} r_i = 0$, o que implica que $det(C) = 0$.

Após ser efetuada a prova do teorema enunciado, este pode ser usado para provar a Total Unimodularidade da matriz das restrições do Problema de Afetação. Tal prova será efetuada de seguida.

Unimodularidade da matriz das restrições do Problema de Afetação

Para exemplificar a unimodularidade da matriz, será considerado o Problema de Afetação descrito na secção 3.2.1 para o caso em que $n = 3$, o processo é análogo para qualquer outro valor de n. As restrições (3) apresentadas na Formulação Matemática do Problema de Afetação podem ser escritas na seguinte forma matricial $AX = 1$, onde X é o vetor constituído pelas variáveis binárias x_{ij} e A é uma matriz com a seguinte estrutura:

$$A = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

Cada uma das colunas da matriz A representa uma variável binária x_{ij}, as primeiras n linhas da matriz A representam as n restrições obtidas em (2.1b) e as últimas n linhas da matriz representam as restrições associadas a (2.1c). Sendo assim, considerando $Q_1 = \{1, ..., n\}$ e $Q_2 = \{n+1, ..., 2n\}$ pelo teorema enunciado anteriormente conclui-se que a matriz de

restrições do Problema de Afetação é Totalmente Unimodular.

Considere-se então o seguinte teorema:

Teorema 3.2.2.2: *Considere-se A uma matriz Totalmente Unimodular e b um vetor inteiro. Então todas as soluções básicas do poliedro* $P' = \{x \geq 0 : Ax = b\}$ *são inteiras.*

Prova: *Dada a matriz A, supõe-se que as colunas da matriz são permutadas. Então obtém-se a partição de A dada por duas matrizes (B,N). A matriz B é não singular (uma matriz singular é uma matriz que não possui inversa) e é designada por matriz base do Problema Linear. Seja* $x = (x_B, x_N)$*, onde* x_B *é o vetor das variáveis básicas associadas às colunas de B e* x_N *o vetor das variáveis não básicas associadas às colunas de N. Então as equações do poliedro* P' *são escritas da seguinte forma:*

$$Bx_B + Nx_N = b \tag{3.4}$$

Visto que B é não singular, pode-se resolver 3.4 obtendo a expressão de x_B *em função de* x_N*:*

$$x_B = B^{-1}b - B^{-1}Nx_N$$

Então, uma solução básica de P' *é:*

$$x_B = B^{-1}b, \qquad x_N = 0$$

Assim uma condição suficiente para que uma solução básica seja inteira é que a matriz B^{-1} *seja uma matriz inteira. Para obter condições para que* B^{-1} *seja uma matriz inteira, introduz-se a noção de matriz unimodular (apenas unimodular e não totalmente unimodular). Uma matriz quadrada inteira B diz-se unimodular se D=*$|detB| = 1$*. Se B é não singular então:*

$$B^{-1} = \frac{B^+}{detB}$$

(Capítulo 2.7 (Garfinkel and Nemhauser, 1972)) onde B^+ *é a matriz adjunta de B (*B^+ *é uma matriz inteira). Então B unimodular implica que* B^{-1} *é uma matriz inteira. Então se A é totalmente unimodular, toda a matriz base B é unimodular e por consequência toda a solução básica* $(x_B, x_N) = (B^{-1}b, 0)$ *é inteira.*

Posto isto, e através do **Teorema 3.2.2.2** prova-se que a resolução do Problema de Afetação considerando a condição de integralidade da solução pode ser transformada na resolução da relaxação linear do problema em questão, cuja solução ótima coincide com a do problema inicial. No **Apêndice B** é apresentado um algoritmo que também pode ser usado para resolver o problema de afetação, nomeadamente o **Algoritmo Húngaro**.

Para finalizar é importante referir que o problema de Afetação representa a formulação do problema em estudo quando não existe restrição para o tamanho de ciclos admissíveis. Isto porque o problema de afetação resume-se a impôr que uma coluna é afetada exatamente com uma linha e vice-versa, sem que duas colunas sejam afetadas com a mesma linha ou vice-versa. O que equivale a impôr que um paciente recebe exatamente um rim e um dador doa exatamente um rim, sem que exista o caso de um dador estar associado a dois pacientes

ou vice-versa. E não existe nenhuma restrição que limite o tamanho dos ciclos formados pelas afetações, logo o emparelhamento entre pares dador/recetor pode ter tamanho ∞.

Sendo assim, este problema será utilizado no algoritmo de pesquisa desenvolvido na Secção 4.2.2, onde este irá representar o majorante da solução a encontrar em cada nó da árvore de pesquisa. Para determinar o valor do majorante, será resolvida a relaxação linear do Problema de Afetação, que como se viu, envolve apenas a solução de um problema linear, para o qual existem algoritmos eficientes.

3.2.3 Transformações efetuadas para utilizar o Problema de Afetação no problema em estudo

O problema em estudo ao longo deste projeto consiste em maximizar o número de transplantes efetuados sem que haja qualquer prioridade/vantagem atribuída à escolha da realização de um transplante entre um dado recetor j e um dado dador i. Por outras palavras, este problema é representado por um grafo dirigido $G(V, E)$, onde os arcos (i, j) possuem peso $w_{ij} = 1$. É necessário também referir que nem todos os recetores são compatíveis com todos os dadores, isto é, não existe um arco (i, j) para todo o vértice i e vértice j existente no grafo.

Para se poder utilizar o problema de afetação para determinar a solução do problema em estudo foi necessário realizar algumas alterações ao grafo inicial G e criar uma matriz de custos com características muito específicas. Primeiramente serão descritas as alterações efetuadas no grafo G e de seguida a forma como foi criada a matriz W. Quanto ao grafo G, foi necessário inserir um arco (i, i) (lacete) com peso 0, isto porque como o dador i e o paciente i são incompatíveis o peso do uso desta aresta na solução final é nulo; aos outros arcos do grafo G foi associado um peso $w_{ij} = 1$ uma vez que, como visto no capítulo 2, maximizar o número de transplantes é equivalente a maximizar o número de arcos selecionados. Na Figura 3.5 é apresentado no lado esquerdo o grafo $G(V, E)$ e no lado direito o grafo modificado, para que seja mais fácil perceber as modificações efetuadas.

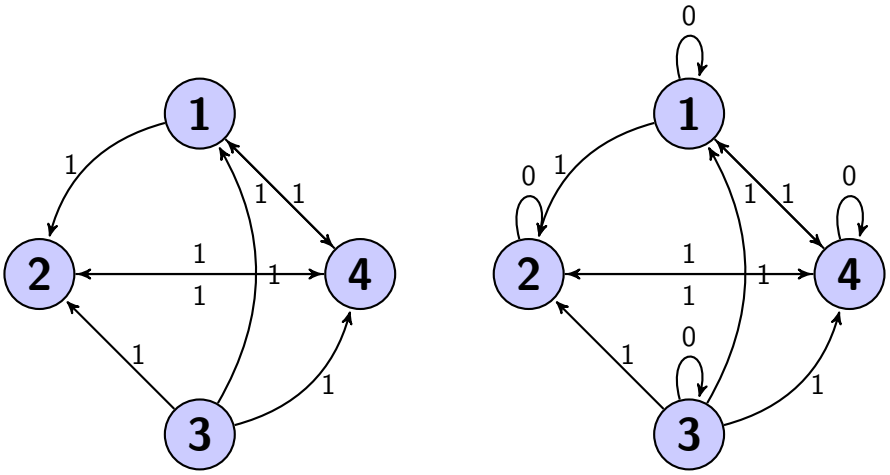


Figura 3.5: Grafo G(V,E) Original e o Grafo G(V,E) Modificado

A matriz de custos W é uma matriz quadrada cujas dimensões são $n \times n$, onde n é o número de vértices do grafo.
O Problema de Afetação é um problema de minimização, enquanto que o problema em estudo é de maximização. Sabe-se que dado um problema de minimização pode-se encontrar a solução do problema de maximização alterando apenas o sinal da função objetivo. De uma forma mais simples, se a função objetivo do problema de minimização é $minf(x)$ ao alterar-se a função objetivo para $-f(x)$ o problema transforma-se num problema de maximização cujas restrições são as mesmas do problema original.

Dadas as propriedades referidas no parágrafo anterior a matriz de custos W foi construída da seguinte forma:
• $w_{ii} = 0 \ \forall$ i $\in$ V;
• $w_{ij} = -1 \ \forall$ (i,j) $\in$ A;
• $w_{ij} = 1000 \ \forall$ (i,j) $\notin$ A;

É importante explicar que a entrada $w_{ij} = 1000$ existe porque como referido anteriormente, o grafo inicial não é um grafo completo, isto é, não existe um arco $(i, j) \forall i, j \in V$ então é necessário associar na matriz W um peso muito elevado a este arco para que ele não seja selecionado (não pode ser selecionado porque não existe). Para que se torne mais clara a construção da matriz W segue-se a matriz de custos (W) correspondente ao grafo apresentado na Figura 3.5:

$$W = \begin{bmatrix} 0 & -1 & 1000 & -1 \\ 1000 & 0 & 1000 & -1 \\ -1 & -1 & 0 & -1 \\ -1 & -1 & 1000 & 0 \end{bmatrix}$$

Todas estas transformações apresentadas acima foram implementadas no algoritmo desenvolvido na secção 4.2.2. Isto porque as instâncias teste para as quais este algoritmo será aplicado inicialmente não estão no formato adequado.

3.3 Algoritmo de Edmonds

O emparelhamento num grafo é um conjunto de arestas que são independentes duas a duas, isto é , quando consideradas duas arestas emparelhadas estas não possuem nenhum vértice em comum. Dado um grafo, um problema bastante conhecido é o de encontrar um emparelhamento com tantas arestas quantas possíveis; existe ainda uma outra versão deste problema, onde as arestas possuem pesos e o desafio é encontrar o emparelhamento que tem o maior peso total. Neste trabalho apenas será abordado o caso em que as arestas não possuem pesos.

Nesta secção será estudado o Algoritmo de Edmonds para o Problema de Emparelhamento de Cardinalidade Máxima; uma descrição detalhada pode ser encontrada em (Papadimitriou and Kenneth, 1998). Primeiramente será estudado o caso do emparelhamento num grafo bipartido, porque este é consideravelmente mais simples e ajuda a ilustrar as ideias básicas

envolvidas na resolução do problema de emparelhamento num grafo geral.

3.3.1 Problema de Emparelhamento e algumas noções básicas

Dada a complexidade do Algoritmo de Edmonds inicialmente serão apresentadas algumas definições e alguns conceitos, de forma a que o algoritmo se torne mais claro e consequentemente de compreensão mais acessível.

Emparelhamento: um emparelhamento M num grafo $G = (V, E)$ é um subconjunto de arestas com a propriedade de que duas arestas de M não possuem nenhum vértice em comum. Considere-se o grafo na Figura 3.6, possíveis emparelhamentos nesse grafo são por exemplo $M_1 = \{[v_2, v_3], [v_4, v_5], [v_6, v_8], [v_7, v_{10}]\}$ e $M_2 = \{[v_1, v_2], [v_3, v_5], [v_4, v_7], [v_6, v_8], [v_9, v_{10}]\}$.

Emparelhamento máximo/perfeito: é um emparelhamento que possui $\lfloor \frac{|V|}{2} \rfloor$ arestas, onde V representa o conjunto dos vértices do grafo. Pode-se verificar que M_2 é o emparelhamento máximo do grafo G, porque possui $\frac{|V|}{2} = 5$ arestas.

Considere-se um grafo G juntamente com um determinado emparelhamento M . As arestas pertencentes a M designam-se por **arestas emparelhadas**; as outras são chamadas **arestas livres**. Se $[v, u]$ é uma aresta emparelhada, então u é o "**par**"de v. Os vértices que não pertencem a nenhuma aresta emparelhada são considerados **vértices expostos**; os outros designam-se **vértices emparelhados**.

Caminho Alternante: dado o caminho $p = [u_1, u_2, ..., u_k]$, diz-se que p é um caminho alternante se as arestas $[u_1, u_2], [u_3, u_4], ..., [u_{k-1}, u_k], ...$ são arestas livres e as arestas $[u_2, u_3], [u_4, u_5], ..., [u_{k-2}, u_{k-1}], ...$ são emparelhadas. Considere-se o grafo G e o emparelhamento M_1 da Figura 3.6. Os caminhos $p_1 = [v_1, v_2, v_3, v_5, v_4, v_8]$ e $p_2 = [v_1, v_4, v_5, v_6, v_8, v_7, v_{10}, v_9]$ são caminhos alternantes.

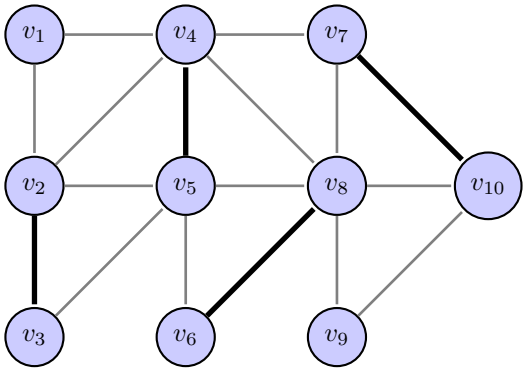


Figura 3.6: Exemplo Ilustrativo Problema de Emparelhamento

Quando existe um caminho alternante cujo vértice inicial é um vértice exposto, um vértice que esteja numa posição ímpar nesse caminho designa-se por **vértice exterior**. Da mesma

forma que um vértice que esteja numa posição par é chamado de **vértice interior**. Devido à existência dos caminhos alternantes anteriormente apresentados, os vértices v_1, v_3, v_4, v_5, v_8 e v_{10} são **vértices exteriores**, porque ocupam uma posição ímpar num destes caminhos e v_1 é um **vértice exposto**.

Caminho aumentante: um caminho alternante $p = [u_1, u_2, ..., u_k]$ diz-se um caminho aumentante se ambos u_1 e u_k são vértices expostos. Na Figura 3.6 o caminho alternante p_2 é um caminho aumentante.

Flor: uma flor é um circuito ímpar que possui $2k + 1$ vértices e k arestas emparelhadas. O exemplo de uma flor é $[u_0, u_1, u_2, u_3, u_4, u_5, u_6]$ no grafo representado na figura 3.7.

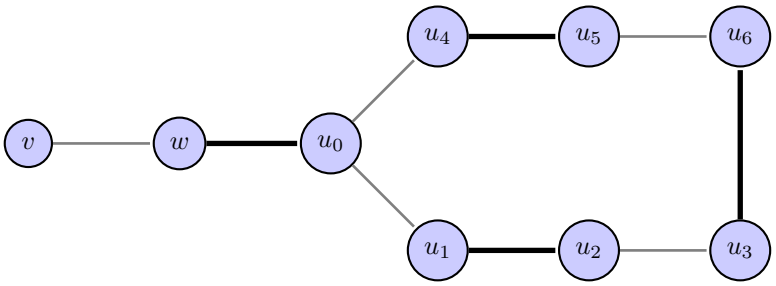


Figura 3.7: Exemplo ilustrativo de uma flor

A importância da existência de caminhos aumentantes para o problema de emparelhamento é expressa no seguinte lema e no seguinte teorema.

Lema 3.3.1.1: *Seja P um conjunto de arestas num caminho aumentante $p = [u_1, u_2, ..., u_k]$ num Grafo G relativamente a um emparelhamento M. Então $M' = M \oplus P$ é um emparelhamento de cardinalidade $|M| + 1$, onde $M' = M \oplus P = (M - P) \cup (P - M)$ representa a diferença simétrica de M e P.*

Prova: *Mostra-se primeiro que $M \oplus P$ é um emparelhamento; desta forma duas arestas pertencentes a $M \oplus P$ não podem ter nenhum vértice em comum. Suponha-se o caso em que duas arestas e, e' pertencentes a $M \oplus P$ possuem um vértice em comum. Visto que, $M \oplus P = (M - P) \cup (P - M)$ existem três possibilidades:*

1. $e, e' \in M - P$

2. $e, e' \in P - M$

3. $e \in M - P$, $e' \in P - M$

No primeiro caso, tem-se duas arestas em M com um vértice em comum, isto é, uma contradição. No segundo caso, note-se que as arestas pertencentes a P-M são arestas da forma $[u_{2j-1}, u_{2j}]$ e portanto duas arestas desta forma não podem ter nenhum vértice em

comum. Para o caso três, suponha-se que uma aresta $e^{'} = [u_{2j-1}, u_{2j}]$ *pertencente a P-M tem um vértice em comum com outra aresta* $e \in M - P$. *Assume-se, sem perda de generalidade, que esse vértice é* u_{2j}. *Mas,* u_{2j} *é o vértice da aresta* $e^{''} = [u_{2j}, u_{2j+1}] \in M$, *e portanto as duas arestas* $e^{''}$ *e e de M têm um vértice em comum, uma contradição. Daí resulta que,* $M^{'}$ *é um emparelhamento. Agora P contém 2k-1 arestas; k dessas arestas são livres (* $[u_1, u_2], [u_3, u_4], ..., [u_{2k-1}, u_{2k}], ...$*) e k-1 fazem parte de M. Portanto* $M^{'} = M \oplus P$ *tem* $|M| + 1$ *arestas.*

Veja-se um exemplo, onde este lema se aplica. Considere-se na Figura 3.6 o caminho aumentante $p_2 = [v_1, v_4, v_5, v_6, v_8, v_7, v_{10}, v_9]$ relativamente ao emparelhamento M_1. Seja $M_3 = M_1 \oplus p_2 = \{[v_1, v_4], [v_2, v_3], [v_5, v_6], [v_7, v_8], [v_9, v_{10}]\}$. É de notar que, M_3 é máximo (tem $\frac{|V|}{2} = 5$ arestas), por isso não faz sentido procurar um outro caminho aumentante em G relativamente a M_3; não pode existir nenhum caminho aumentante associado ao emparelhamento máximo, uma vez que tal caminho poderia ser usado pelo **Lema 3.3.1.1** para aumentar o emparelhamento atual. O inverso desta afirmação também é válido como se pode constatar no seguinte teorema:

Teorema 3.3.1.1: *Um emparelhamento M num Grafo G é máximo se e só se não existir nenhum caminho aumentante em G relativamente a M.*

Prova: *Uma direção da prova resulta diretamente do* ***Lema 3.3.1.1*** *. Para a outra direção supõe-se que não existe nenhum caminho aumentante em G relativamente a M, e que M não é máximo. Deste modo, existe um emparelhamento* $M^{'}$ *em G tal que* $|M^{'}| > |M|$. *Considerem-se as arestas em* $M \oplus M^{'}$*; estas arestas formam um subgrafo de G (que pode ser desconexo). Visto que duas arestas de um emparelhamento não podem ter um vértice em comum o subgrafo* $G^{'} = (V, M \oplus M^{'})$ *tem uma estrutura especial - todos os vértices têm grau* $\leq$ *2. Se o grau de um vértice é 2, uma das arestas pertence a M e a outra a* $M^{'}$. *Portanto, todas as componentes conexas de* $G^{'}$ *serão ou caminhos ou circuitos de comprimento par. Em todos os circuitos tem-se o mesmo número de aresta de M e de* $M^{'}$. *Pois* $|M^{'}| > |M|$, *tem de ser o caso em que um dos caminhos tem mais arestas de* $M^{'}$ *do que de M e por isso este caminho é um caminho aumentante. Contudo, isto contradiz a hipótese de que não existe nenhum caminho aumentante em G relativamente a M, e o teorema está provado.*

A descrição detalhada do algoritmo de Edmonds para grafos bipartidos encontra-se no **Apêndice C.1**. Enquanto que a definição de flores e a explicação do algoritmo de Edmonds para grafos gerais são apresentadas nos **Apêndices C.2** e **C.3**, respetivamente.

Capítulo 4

Trabalho Desenvolvido

Neste capítulo é apresentado todo o trabalho desenvolvido ao longo desta tese. Primeiramente descreve-se detalhadamente uma ferramenta criada para a ASST para determinar a solução do problema de Transplantação Renal Emparelhada para um determinado conjunto de pares dador/recetor. De seguida, existe uma secção onde é apresentado um exemplo de aplicação do algoritmo desenvolvido, e uma descrição e análise do mesmo.

4.1 Ferramenta de apoio à decisão para o PNDRC

Nesta secção irá apresentar-se, e explicar-se detalhadamente, uma ferramenta criada no âmbito do PNDRC para determinar a solução do problema KEP para um determinado conjunto de pares, isto é, dado um conjunto de pares dador/recetor incompatíveis criar um novo conjunto de pares compatíveis, segundo algumas condições, e nesse novo conjunto maximizar o número de transplantes efetuados utilizando o conceito de Transplantação Renal Emparelhada. Esta ferramenta foi usada pela ASST-Autoridade para os Serviços de Sangue e da Transplantação nas três últimas reuniões em que se procurou encontrar pares compatíveis no âmbito do PNDRC.

No esquema seguinte estão representadas as quatro etapas que fazem parte do processo de obtenção da solução:

Figura 4.1: Esquema das etapas que fornecem a solução

4.1.1 Base de Dados

Inicialmente foi necessário organizar os dados relativos a cada par numa base de dados elaborada em Excel. Nessa base de dados estão armazenadas as seguintes características para cada um dos dadores e dos recetores:

Nº Identificação	**Data início diálise**
Nome	**Nrº meses de diálise**
Residente	**Situação Clínica**
Profissão	**Confirmação situação clínica**
Data de Nascimento	**Data Atualização dos dados**
Idade	**Altura**
Género	**Peso**
Naturalidade	**IMC**
Hospital	**Prova Esforço**
Nº Processo	**Grupo Sanguíneo**
Relação Parental	**Doença Renal**
Rx Torax	**ECG**
ECG	**Estudo Laboratorial**
HLA(A,B,DR)	**Alossensibilização**
Especificidades HLA	**PRA Histórico**
PRA Atual	

Tabela 4.1: Características dos dadores e dos recetores

É de notar que naturalmente existem algumas características que apenas possuem informação para um dos elementos do par. Por exemplo, apenas os recetores têm informação sobre a doença renal e o número de meses de diálise, assim como apenas existe informação sobre o peso e a altura para os recetores.

Após os dados estarem todos organizados e recorrendo à aplicação VBA existente no Excel (Buyens, 2003) (Nina, 1999), foram programadas duas filtragens diferentes.

A primeira filtragem designou-se por "filtragem dos pares dador/recetor com compatibilidade sanguínea"e foi baseada nos tipos de sangue de cada dador e na tabela 1.1 apresentada no capítulo 1.

Esta filtragem teve como objetivo encontrar um novo conjunto de pares dador/recetor que possuissem compatibilidade sanguínea. Para cada recetor foram encontrados todos os dadores disponíveis na base de dados que possuiam um grupo sanguíneo compatível com o do recetor em causa. De cada vez que se encontrava um dador compatível era criado um novo par dador/recetor constituído pelo recetor e pelo dador em questão.

Para que seja mais fácil perceber o funcionamento e o resultado final desta filtragem, apresenta-se de seguida um exemplo ilustrativo.

Considerem-se os pares representados na Figura 4.2 :

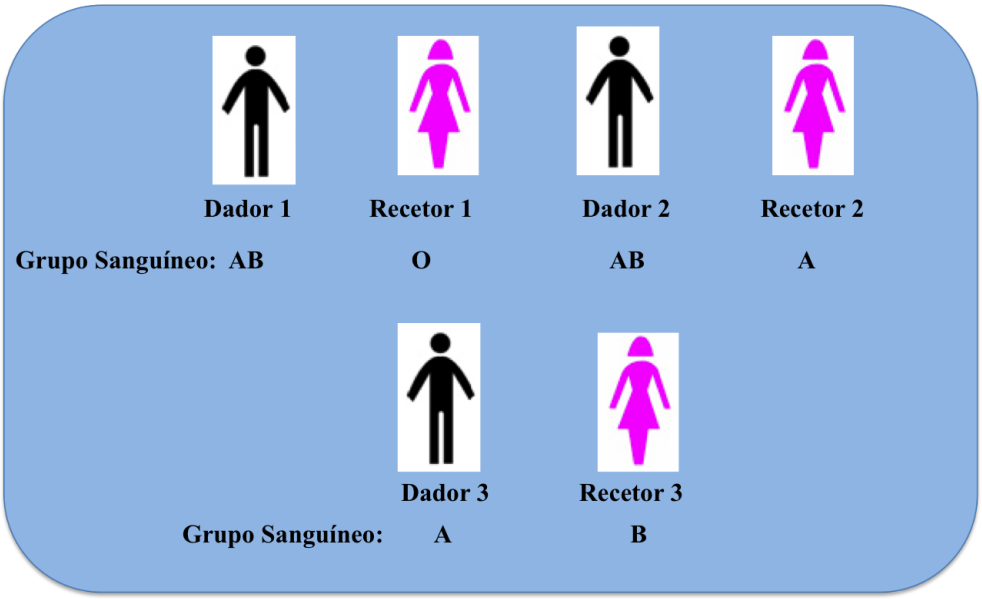


Figura 4.2: Pares dador/recetor incompatíveis

Todos estes pares são incompatíveis, isto é, dado um par o dador e o recetor correspondentes a esse par são incompatíveis. Mas, se for aplicada a 1ª filtragem obtém-se um novo par compatível formado pelo dador do par 3 e pelo recetor do par 2.

Após obter os resultados da primeira filtragem, pode-se aplicar a segunda. A segunda filtragem designou-se por "teste do crossmatch"e apenas se aplica aos pares obtidos na 1ªfiltragem. Já que após serem formados os pares compatíveis relativamente ao tipo de sangue, é necessário verificar se possuem crossmatch positivo ou negativo.

O teste do crossmatch consiste em dadas as especificidades que um recetor possui para cada uma das componentes do sistema HLA, verificar se alguma delas coincide com os elementos do sistema HLA do dador. De forma mais simples, cada pessoa tem 2 valores para cada um dos 3 tipos de HLA (HLA A, HLA B, HLA DR) e contém ainda especificidades para cada um destes. Se um dado recetor possuir uma especificidade para o HLA tipo A igual a um dos elementos do HLA tipo A do seu dador o teste do crossmatch será positivo e este par será incompatível apesar de possuir compatibilidade sanguínea. O mesmo acontece com o HLA tipo B e o HLA tipo DR.

Mais uma vez, segue-se um exemplo para que se torne mais clara toda a informação relatada sobre a 2ªfiltragem.

Considerem-se os 2 pares representados na Figura 4.3, obtidos após a aplicação da 1ªfiltragem:

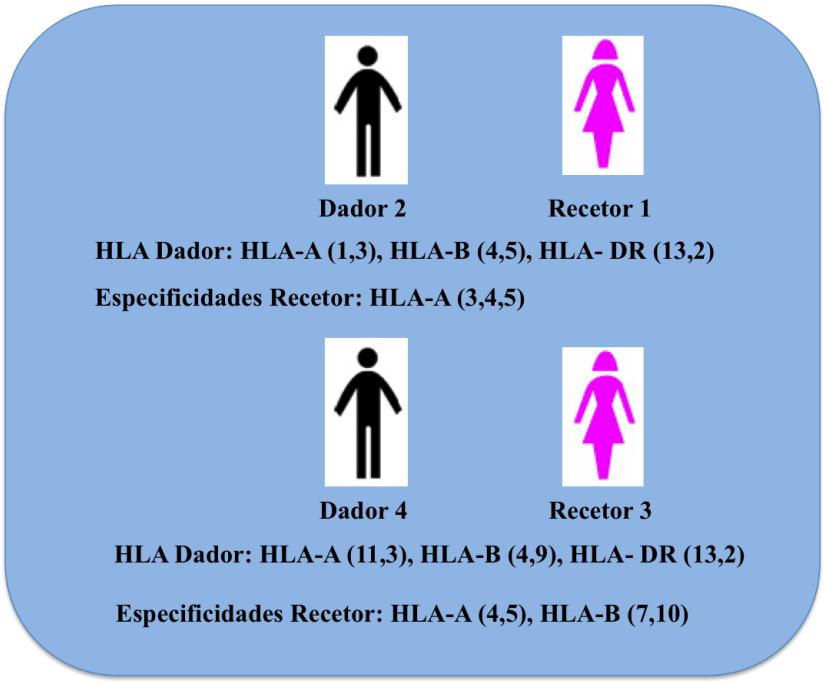


Figura 4.3: Sistema HLA e especifidades par dador/recetor

Aplicando a segunda filtragem a estes dois pares o resultado final seria apenas o par formado pelo dador 4 e pelo recetor 3, uma vez que o recetor 1 possui especificidade contra o HLA-A 3 que é um dos HLA-A do dador 2.

A base de dados, as duas filtragens efetuadas e os outputs necessários para introduzir no CPLEX foram todos programados, como referido anteriormente, através da aplicação VBA do Excel. Na base de dados final para obter cada uma das filtragens ou cada um dos outputs apenas é necessário selecionar um botão que possui o nome igual ao da filtragem que se pretende efetuar, pois a esse botão está associada toda a programação necessária para a realização da ação em questão. O interface final pode ser visualizado na seguinte imagem:

Figura 4.4: Interface Base de Dados

4.1.2 Otimização

Os pares dador/recetor que resultam da aplicação das duas filtragens são aqueles que têm compatibilidade sanguínea e imunológica (crossmatch negativo) e formam um grafo no qual um arco (i,j) significa que o dador i é compatível com o recetor j.

Dado este grafo foi criado um programa baseado na Formulação Ciclo que será executado no otimizador para programação inteira CPLEX (IBM ILOG, Inc, 2012) cujo output corresponde aos transplantes que se podem efetuar usando a Transplantação Renal Emparelhada considerando um tamanho de ciclo pré-definido.

4.2 Algoritmo Desenvolvido

O algoritmo desenvolvido pretende resolver o seguinte problema: dado um conjunto de pares dador/recetor maximizar o número de transplantes realizados para ciclos de tamanho máximo 3.

A ideia base deste algoritmo consiste em: em cada iteração selecionar um ciclo de tamanho 3, de entre todos os possíveis, e afetar os restantes pares usando o algoritmo de Edmonds (ciclos de tamanho 2). Tendo em conta que a solução encontrada pelo algoritmo de Edmonds é um minorante do problema, com a inclusão de ciclos de maior dimensão pretende-se encontrar uma solução que resulte num número de transplantes superior.

Primeiramente será apresentado um exemplo de aplicação do algoritmo para que depois seja mais fácil acompanhar a descrição teórica deste. No final da descrição teórica será apresentada uma análise detalhada do algoritmo desenvolvido.

4.2.1 Exemplo de Aplicação do Algoritmo Desenvolvido

Para que se torne mais claro o funcionamento do algoritmo, é apresentado um exemplo de aplicação. Considere-se o digrafo representado na Figura 4.5.

Inicialização

O primeiro passo é criar o nó raiz. Para se puder aplicar a função do algoritmo de Edmonds existente no *python* (função **max_weight_matching** do pacote **networkx.algorithms.matching**) é necessário primeiramente construir o grafo não dirigido correspondente ao digrafo em questão. Este grafo é construído da seguinte forma: se existem os arcos (i, j) e (j, i) no digrafo então existe a aresta (i, j) no grafo não dirigido. Para o exemplo em questão o grafo não dirigido resultante pode ser visualizado no lado direito da Figura 4.5.

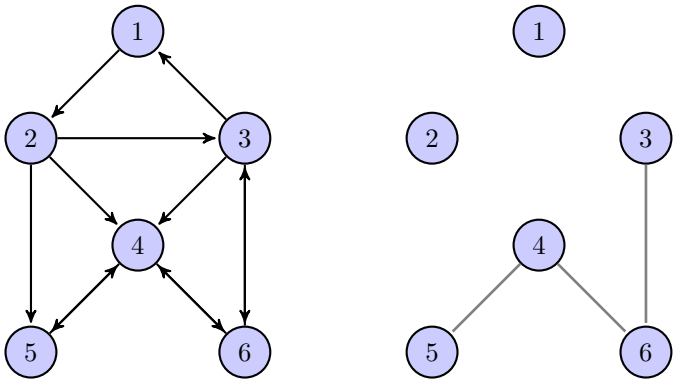


Figura 4.5: Exemplo de Aplicação do Algoritmo Desenvolvido

Posto isto determina-se a solução de Edmonds que neste caso são as arestas $[3, 6], [5, 4]$ que correspondem aos arcos $[(3, 6), (4, 5), (5, 4), (6, 3)]$. Depois determinam-se os ciclos de tamanho 3 disponíveis na raiz, $[[1, 2, 3], [3, 4, 6]]$, o valor do majorante (através do algoritmo húngaro) 6 e os arcos correspondentes $[(1, 2), (2, 5), (5, 4), (4, 6), (6, 3), (3, 1)]$. Existem ainda

duas listas nomeadamente, a lista ciclos_usados e a lista ciclos_avaliados. Para cada nó na árvore de pesquisa, como os próprios nomes indicam, a primeira lista armazena os ciclos disponíveis e a segunda os ciclos usados. Para o nó raiz estas listas corrrespondem ao conjunto vazio.

Toda a informação relevante associada a cada nó é guardada numa lista de espera D. Os nós que pertencem a D são nós que ainda não foram analisados. A ordem de inserção dos nós em D é feita consoante o tipo de pesquisa que se esteja a utilizar. Sendo assim insere-se o nó raiz em D, com toda a informação relevante que lhe está associada.

No final desta etapa D apenas possui a informação relativa ao nó raiz (ver Figura 4.6). E a árvore de pesquisa é representada apenas por um nó (Figura 4.6).

D = [solução de Edmonds=[[3, 6], [5, 4]], majorante=6, arcos do majorante= $[(1,2),(2,5),(3,1),(4,6),(5,4),(6,3)]$, ciclos_disponiveis= $[[1,2,3],[3,4,6]]$, ciclos_usados=[ ],ciclos_avaliados=[ ]].

Figura 4.6: Árvore de Pesquisa inicial

Pesquisa em Árvore

Em cada iteração da pesquisa em árvore é selecionado um nó pertencente a D. Para o exemplo em estudo será utilizada pesquisa em profundidade, como tal será sempre selecionado o último nó de D (a ordem de inserção dos nós será apresentada posteriormente). Para o nó selecionado analisa-se se este não verifica nenhuma das condições que levam à interrupção da pesquisa ou da exploração do nó. No caso do nó não verificar nenhuma condição, seleciona-se um ciclo de tamanho 3 e criam-se dois ramos: ramo direito e ramo esquerdo. O ramo esquerdo representa a utilização do ciclo de tamanho 3 selecionado, enquanto que o ramo direito está associado à situação em que não se utiliza o ciclo.

1ª iteração

Na 1ª iteração o nó selecionado corresponde ao nó raíz. Este é analisado e atualiza-se a melhor solução atual. Uma vez que este passo não verificou nenhuma condição que levasse à interrupção da pesquisa ou da exploração do nó, seleciona-se um ciclo de tamanho 3 para ser analisado, por exemplo $[1, 2, 3]$, e constroem-se os dois ramos correspondentes. O ramo direito d_1, possui a mesma solução de Edmonds que o nó que está a ser explorado, ou seja, a solução é $[(3,6),(4,5),(5,4),(6,3)]$; o valor do majorante 6 e os arcos correspondentes ao majorante são $[(1,2),(5,4),(4,6),(3,1),(6,3),(2,5)]$, como se pode visualizar na Figura 4.7.

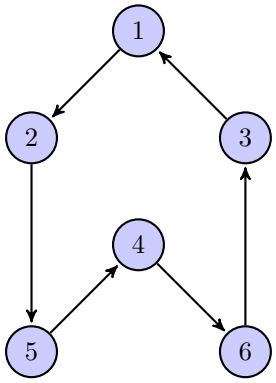


Figura 4.7: Digrafo correspondente ao majorante do ramo d_1

O ciclo disponível para este ramo é $[[3, 4, 6]]$. Isto porque é o único que estava disponível no nó em análise, após retirar o que foi selecionado. A lista ciclos_usados continua vazia porque esta lista corresponde aos ciclos usados na solução constituída por ciclos de tamanho 2 e 3 e o ramo direito está associado ao caso em que não se usa o ciclo de tamanho 3. A lista ciclos_avaliados passa a possuir o ciclo $[1, 2, 3]$. Por fim, insere-se o ramo direito em D.

Na construção do ramo esquerdo e_1, o primeiro passo consiste em remover do digrafo atual todos os arcos que possuam um vértice em comum com o ciclo selecionado. Neste caso o digrafo fica resumido aos arcos $[(4, 6), (6, 4), (4, 5), (5, 4)]$ (ver Figura 4.8) uma vez que todos os outros possuem um vértice em comum com o ciclo selecionado.

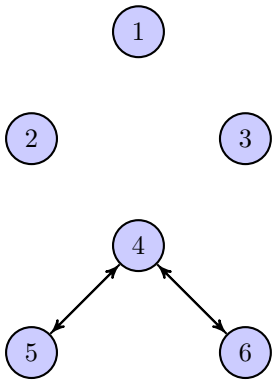


Figura 4.8: Digrafo correspondente ao ramo e_1

Seguidamente, constrói-se o grafo não dirigido associado ao novo digrafo e encontra-se a solução de Edmonds para este grafo, $[(4, 6), (6, 4)]$. O novo valor do majorante é 5 e os arcos correspondentes são $[(5, 4), (4, 5)]$ e $[[1, 2, 3]]$ (Figura 4.9). Verifica-se que não existe mais nenhum ciclo disponível para este nó, ou seja, ciclos_disponíveis=[]. Atualizam-se as listas ciclos_usados e ciclos_avaliados. Por último, insere-se o ramo esquerdo e_1 em D.

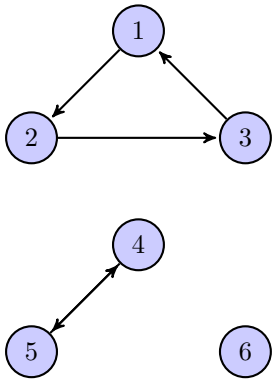


Figura 4.9: Digrafo correspondente ao majorante do ramo e_1

No final da primeira iteração a fila D é constituída por d_1 e e_1, e a informação relevante armazenada em cada nó é: d_1: [solução de Edmonds=[[3, 6], [5, 4]], majorante=6, arcos do majorante= $[(1,2),(2,5),(3,1),(4,6),(5,4),(6,3)]$, ciclos_disponiveis=[[3, 4, 6]], ciclos_usados=[ ], ciclos_avaliados=[[1, 2, 3]]], e_1: [solução de Edmonds=$[(4,6),(6,4)]$, majorante=5, arcos do majorante= $[(5,4),(4,5)]$ e [[1, 2, 3]], ciclos_disponiveis=[], ciclos_usados=[[1, 2, 3]], ciclos_avaliados=[[1, 2, 3]]]. E a árvore de pesquisa é constituída por 3 nós como se pode verificar na Figura 4.10.

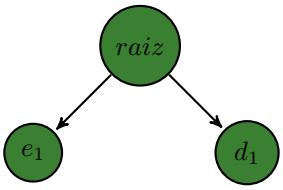


Figura 4.10: Árvore de Pesquisa associada à 1ª iteração

2ª iteração

Volta-se ao início do passo do algoritmo em que se seleciona outro nó para explorar; esse nó corresponde ao último inserido em D, neste caso e_1. Ao analisar-se a informação referente a este nó, atualiza-se a melhor solução atual e verifica-se que este não possui ciclos de tamanho superior a quatro no digrafo constituído pelos arcos na solução do majorante. Posto isto, este nó verifica a condição que faz com que não valha a pena explorar o ramo em questão, e_1, e o algoritmo regressa à seleção de um novo nó em D.

3ª iteração

O próximo nó a ser selecionado, neste caso, corresponde ao ramo d_1. Sendo assim e dado que este não verificou nenhuma condição para interromper a pesquisa ou a exploração do nó, seleciona-se o ciclo a ser avaliado; este só pode ser $[3, 4, 6]$ (pois não existe mais nenhum disponível). Seguindo o raciocínio anterior a informação referente ao ramo direito d_2 é: solução de Edmonds $[(3,6),(4,5),(5,4),(6,3)]$, majorante 6 e arcos correspondentes $[(1,2),(5,4),(4,6),(3,1),(6,3),(2,5)]$, ciclos disponíveis =[ ]. Atualizam-se novamente as

listas ciclos_usados e ciclos_avaliados e insere-se o ramo direito em D.

Da mesma forma no caso do ramo esquerdo e_2, o digrafo correspondente a este ramo passa a ser $[(1,2),(2,5)]$ (ver figura 4.11(a)), o valor do majorante é 3 e os arcos correspondentes são $[3,4,6]$ (ver figura 4.11(b)); a solução de Edmonds é [ ], e não há ciclos disponíveis. Insere-se o ramo e_2 em D e regressa-se à seleção de um novo nó.

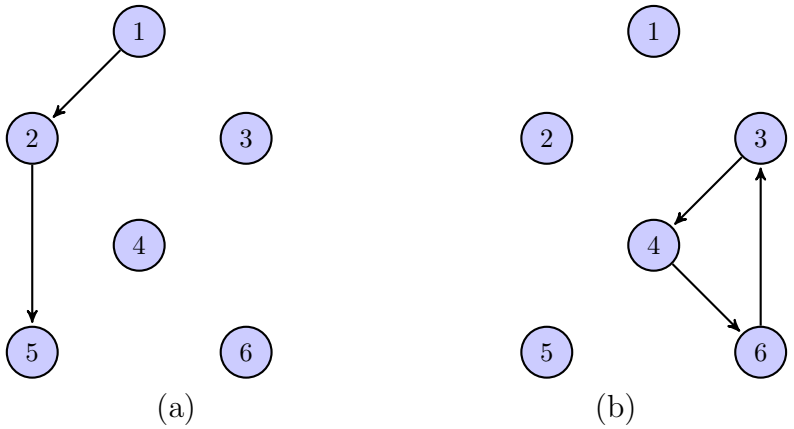


Figura 4.11: Digrafo correspondente a e_2 e digrafo correspondente ao seu majorante

No final da terceira iteração $D = [e_2, d_2]$. Para d_2 a informação armazenada é solução de Edmonds=$[(3,6),(5,4)]$, majorante=6, arcos do majorante= $[(1,2),(2,5),(3,1),(4,6),(5,4),(6,3)]$, ciclos_disponiveis=[], ciclos_usados=[ ], ciclos_avaliados=$[[1,2,3],[3,4,6]]$. No caso do ramo e_2 a informação associada a este nó foi solução de Edmonds=[], majorante=3, arcos do majorante= $[3,4,6]$, ciclos_disponiveis=[], ciclos_usados=$[[3,4,6]]$, ciclos_avaliados=$[[1,2,3],[3,4,6]]$.

A árvore de pesquisa obtida no final desta iteração está representada na Figura 4.12.

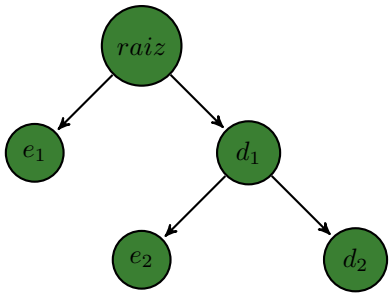


Figura 4.12: Árvore de Pesquisa associada à 3ª iteração

4ª iteração

O próximo nó a ser selecionado é o ramo esquerdo acabado de construir, nomeadamente e_2; uma vez que este nó possui um valor do majorante inferior à melhor solução encontrada,

este ramo é cortado e volta-se a selecionar um novo nó.

5ª iteração

Neste momento, existe um único nó que não possui mais nenhum ciclo de tamanho três para analisar, por isso a sua exploração termina, e consequentemente a pesquisa para D=[ ].
Por último retorna-se a melhor solução encontrada: [(4, 6), (6, 4)][[1, 2, 3]].

A árvore de pesquisa resultante para este exemplo pode ser visualizada na Figura 4.13, apresentada de seguida:

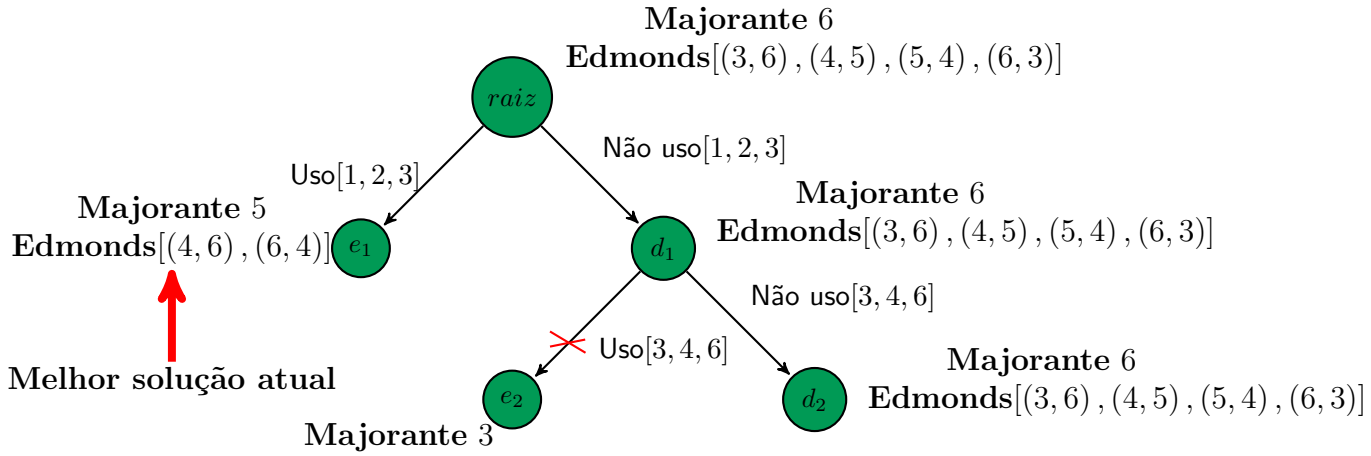


Figura 4.13: Árvore de Pesquisa final

4.2.2 Descrição de Algoritmo

Ao longo desta secção será descrito o algoritmo criado neste trabalho para o problema de maximização em estudo detalhando cada um dos seus passos. Também será apresentado o pseudocódigo e um fluxograma do algoritmo.

O algoritmo desenvolvido neste trabalho consiste num algoritmo de pesquisa em árvore binária, onde são abordados os dois tipos de pesquisa (largura e profundidade). Como referido anteriormente, a única diferença entre a pesquisa em largura e a pesquisa em profundidade é o modo como é selecionado o nó que será analisado no passo seguinte. Posto isto, será apenas descrito todo o procedimento do algoritmo para um tipo de pesquisa, uma vez que este é análogo para o outro tipo.

O objetivo deste algoritmo é a determinação de uma afetação entre pares dador/recetor para o problema em questão, constituída apenas por ciclos de tamanho 2 ou 3. Este algoritmo começa por encontrar um emparelhamento que considere só ciclos de tamanho 2, usando o algoritmo de Edmonds. Esta solução corresponde à raiz da árvore de pesquisa, isto é, é a partir da solução constituída apenas por emparelhamentos de tamanho 2 que a

pesquisa se inicia. Depois seleciona um ciclo de tamanho 3 existente no digrafo associado ao problema e analisa se este ciclo pode melhorar a solução atual. Esta análise é feita através de determinadas condições impostas que mais tarde serão descritas detalhadamente. No caso de se verificar que vale a pena analisar o ciclo em questão, são criados dois ramos associados a esse ciclo, "Ramo Esquerdo"e "Ramo Direito". O "Ramo Esquerdo"corresponde à situação em que se decide usar o ciclo selecionado e o "Ramo Direito"corresponde ao caso em que não se usa o ciclo. O algoritmo avança selecionando iterativamente os diversos ciclos de tamanho 3 existentes no algoritmo e termina quando não existir mais nenhum ciclo para analisar ou então caso se encontre uma solução que se prove ser ótima, isto é, o número de transplantes desta solução é igual ao majorante do nó raiz.

Regras de poda de nós e ramos

Neste algoritmo existem duas regras para interromper a exploração de um nó, uma regra para parar a pesquisa e uma outra regra para interromper a exploração de um ramo (para cortar um ramo). Na descrição do algoritmo apresentada de seguida será considerada a versão do algoritmo com pesquisa em profundidade.

Relativamente à regra para interromper a exploração de um ramo, esta considera o facto do majorante para o nó que vai ser analisado nesse ramo ser inferior ou igual ao tamanho da melhor solução encontrada até ao momento. Isto é, para cada nó que vai ser analisado é calculado um majorante (através do problema de afetação) para a solução desse nó. Se o valor desse majorante não for superior à melhor solução atual, conclui-se que não vale a pena explorar aquele ramo e portanto esse ramo é podado.

Relativamente às regras para interromper a exploração de um nó, a primeira regra utiliza os arcos pertencentes à solução do problema resolvido para encontrar o majorante do nó através da resolução de um problema de afetação. Se no digrafo constituído por esses arcos não existir nenhum ciclo de tamanho igual ou superior a quatro, então não vale a pena explorar o nó em questão. A segunda regra considera que, se num dado momento da pesquisa não existir mais nenhum ciclo de tamanho três para analisar, a pesquisa naquele nó pode terminar e portanto este nó não é mais explorado.

Regra de paragem

A regra existente para parar a pesquisa é a seguinte: para o nó raiz é calculado um minorante pelo algoritmo de Edmonds e um majorante pelo problema de afetação. O valor obtido pelo problema de afetação é um majorante para a solução ótima que se vai encontrar ao longo da pesquisa. Portanto, se num dado nó o valor da solução encontrada coincidir com o majorante do nó raiz, a solução ótima foi determinada e a pesquisa termina.

Na exploração de cada nó a melhor solução atual vai sendo atualizada. Por outras palavras, a melhor solução atual na raiz é a solução de Edmonds, mas se ao longo da pesquisa num dado nó for encontrada uma solução de tamanho superior (número de transplantes realizados superior) é necessário atualizar a melhor solução, que passa a ser a solução do nó que está a ser analisado.

O pseudocódigo do algoritmo é descrito na Figura 4.14 e um fluxograma onde está resumido o algoritmo é apresentado no final da descrição do algoritmo (Figura 4.15). É importante

referir que foi necessária a construção de pequenas funções básicas: uma função para determinar os ciclos de tamanho três existentes no grafo, uma função que remove um determinado ciclo ao digrafo atual, uma função que constrói o grafo não dirigido associado ao digrafo atual e por último uma função que verifica se existem ciclos de tamanho superior a 3 no digrafo constituído pelos arcos do majorante.

O input do algoritmo é um digrafo $G(V, E)$, onde $v \in V$ representa um par dador/recetor incompatível e um arco $(i, j) \in E$ traduz que o dador i pode doar o seu rim ao recetor j.
O primeiro passo consiste na criação do nó raiz; como o input do algoritmo é um grafo dirigido e a função existente para determinar a solução de Edmonds tem como input um grafo, é necessário primeiramente construir o grafo associado ao digrafo inicial. Essa construção é feita através de uma função que dado um digrafo retorna as arestas que constituem o grafo correspondente. Isto é, se existirem os arcos (i,j) e (j,i) no digrafo irá existir a aresta (i,j) no grafo. Após a obtenção do grafo G, determina-se a solução de Edmonds correspondente. De seguida calcula-se o majorante e os arcos correspondentes através da resolução do Problema de Afetação para o digrafo G e por último encontram-se todos os ciclos de tamanho três existentes no digrafo. Em cada nó existem ainda duas listas, ciclos_usados e ciclos_avaliados através das quais é possível saber em cada etapa da pesquisa quais os ciclos de tamanho 3 usados e quais os que foram avaliados. Obviamente que para o nó raiz estas duas listas correspondem a listas vazias. O nó raiz já possui toda a informação necessária, e pode então ser inserido na fila D, onde ao longo da pesquisa serão inseridos os outros nós.

Enquanto a fila D não estiver vazia, seleciona-se e remove-se o último elemento da fila. Isto porque se trata de pesquisa em profundidade e a ordem de inserção dos nós na fila é primeiro o nó referente ao ramo direito e depois o nó referente ao ramo esquerdo.
Uma vez selecionado o nó, primeiramente analisa-se se este verifica alguma das condições que faz com que o ramo seja cortado ou o nó em questão seja abandonado por não valer a pena ser explorado. Caso isto aconteça a pesquisa avança para outro nó. Pode ainda acontecer a situação de que o nó que está a ser analisado contenha a solução ótima. Neste caso a pesquisa para. Se não se verificar nenhuma das situações anteriores a exploração do nó avança selecionando-se um ciclo de tamanho três disponível e através deste cria-se o ramo esquerdo e o ramo direito correspondentes ao nó em questão.

O nó associado ao ramo direito, como referido anteriormente, diz respeito à situação em que não se usa o ciclo selecionado, portanto o majorante e os arcos correspondentes, a solução do emparelhamento e os ciclos_usados deste nó são iguais às do nó pai. Apenas é necessário atualizar a lista de ciclos_avaliados inserindo-lhe o ciclo selecionado e determinar novamente os ciclos de tamanho três disponíveis. Mas desta vez não é necessário recorrer à função que determina os ciclos de tamanhos 3 existentes no digrafo porque, sabe-se que os ciclos de tamanho 3 disponíveis neste nó serão os ciclos do nó pai exceto os que já foram avaliados (os que pertencem à lista ciclos_avaliados). Portanto removem-se os ciclo_avaliados aos ciclos disponíveis no nó explorado e obtém-se exatamente o conjunto de ciclos pretendidos. Por fim, o nó associado ao ramo direito é inserido em D.

Input: Digrafo G(V,E)

Output: Emparelhamento Máximo constituído por ciclos de tamanho não superior a 3

Passo 1: (construção do nó raiz)
- Obter o grafo correspondente ao digrafo inicial;
- Determinar um minorante para a solução através do algoritmo de Edmonds;
- Encontrar todos os ciclos de tamanho 3;
- Determinar o majorante e os arcos correspondentes através do problema de afetação;
- Inserir o nó raiz em D, onde D representa a fila de espera dos nós a serem analisados;

Passo 2: Enquanto $D \neq \oslash$
- Selecionar e remover o último nó de D;
- Se o majorante $\leq$ tamanho da melhor solução:
  - corta-se o ramo; volta-se ao início do passo 2;
- Se o tamanho da solução $\geq$ tamanho da melhor solução:
  - Atualiza-se a melhor solução atual;
- Se o tamanho da solução atual= majorante do nó raiz:
  - A pesquisa para: encontrou-se a solução ótima;
- Se não existirem ciclos de tamanho ≥ 4 no digrafo constituído pelos arcos do majorante:
  - Não se explora o nó; regressa-se ao início do passo 2;
- Se existirem ciclos de tamanho 3 disponíveis:
  - Seleciona-se um ciclo para avaliar e criam-se os respetivos ramos:

  - **Ramo Direito**
  - majorante e arcos correspondentes, solução de Edmonds, ciclos_usados iguais ao nó pai;
  - Os ciclos disponíveis = remover os ciclos_avaliados aos ciclos disponíveis do nó em análise;
  - Insere-se o ciclo selecionado na lista ciclos_avaliados;
  - Insere-se o Ramo Direito em D;

  - **Ramo Esquerdo**
  - Remove-se o ciclo selecionado ao digrafo atual;
  - Calcula-se a nova solução de Edmonds;
  - Calcula-se o novo valor do majorante e os arcos correspondentes;
  - Insere-se o ciclo selecionado às listas ciclos_usados e ciclos_avaliados;
  - Calcula-se o novo conjunto de ciclos disponíveis;
  - Insere-se o Ramo esquerdo em D.
- caso contrário:
  - Não existem mais ciclos para avaliar; volta-se ao início do passo 2;

Figura 4.14: Pseudocódigo do Algoritmo

O ramo esquerdo, como já foi referido, corresponde ao caso em que o ciclo selecionado é

usado. O primeiro passo é remover o ciclo selecionado ao digrafo atual (são removidos todos os arcos que possuam um vértice em comum com o ciclo selecionado). Após esta remoção calcula-se novamente o majorante e os arcos correspondentes, sendo que no caso de não se tratar do nó raiz o majorante é dado pela solução do Problema de Afetação para o digrafo atual (de onde são excluídos todos os vértices que se encontram em ciclos de tamanho 3 entretanto selecionados) mais o número de transplantes associados aos ciclos de tamanho 3 selecionados. Análogamente os arcos do majorante são os arcos da solução do Problema de Afetação mais os arcos dos ciclos de tamanho três usados. É também necessário calcular os ciclos de tamanho três disponíveis no digrafo e atualizar as listas ciclos_usados e ciclos_avaliados. Por último, calcula-se a nova solução de Edmonds para o grafo resultante após a remoção dos vértices associados ao ciclo selecionado. Após efetuar todas as alterações necessárias o ramo esquerdo é inserido na fila D.

Todo este processo é realizado repetidamente até que se encontre a solução ótima ou então até que D seja uma lista vazia. No final, é retornado o número de nós explorados a solução obtida (os emparelhamentos de tamanho dois e os ciclos de tamanho três utilizados).

Na Figura 4.15, apresenta-se o fluxograma referente ao algoritmo desenvolvido.

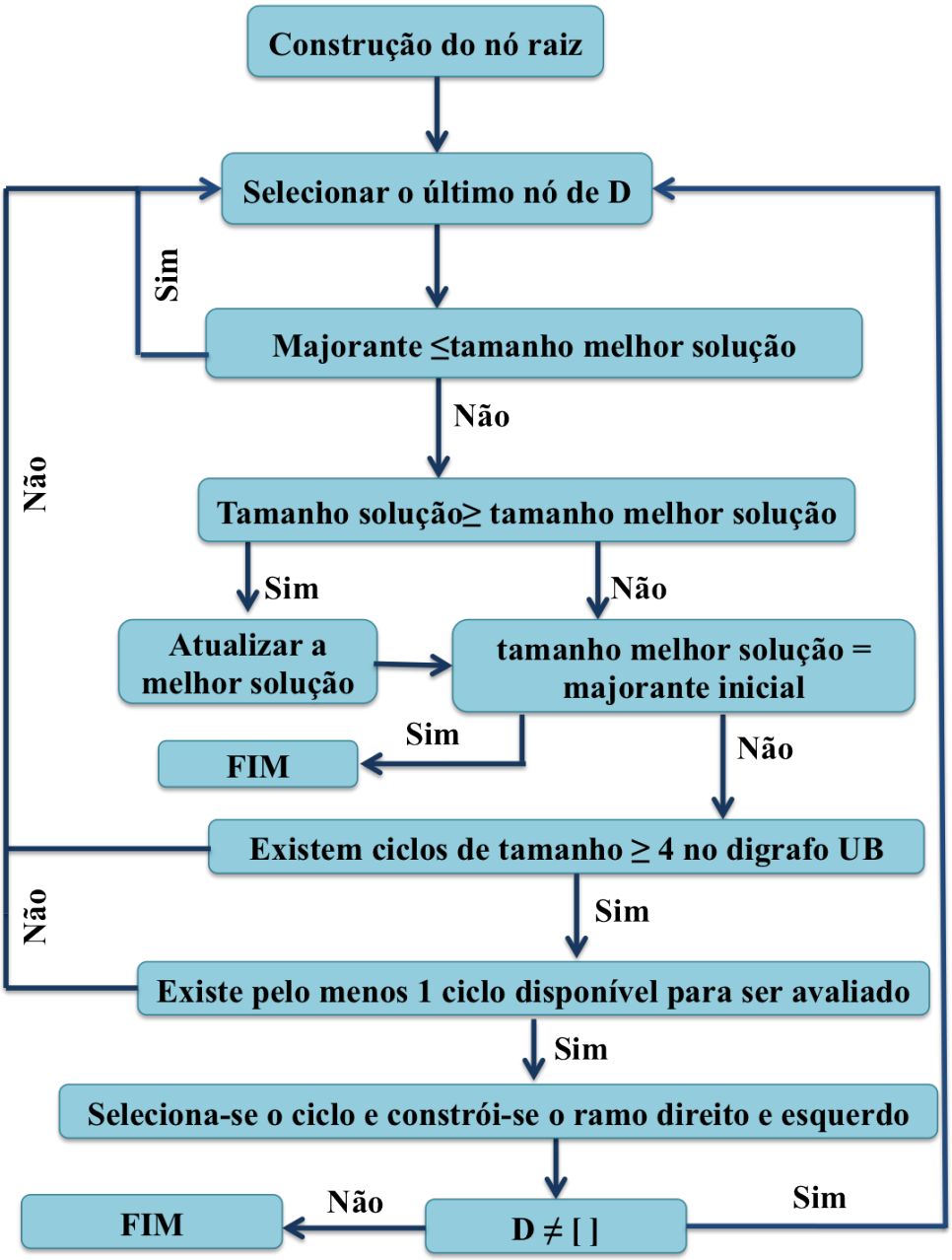


Figura 4.15: Fluxograma do Algoritmo

4.2.3 Análise do Algoritmo Desenvolvido

Quando foram efetuados os primeiros testes ao algoritmo com as instâncias referidas na Secção 5.1 verificou-se que do ponto de vista de CPU necessário este algoritmo era pior relativamente à Formulação Ciclo implementada e resolvida através do software *Gurobi* (formulação com a qual serão comparados os resultados). Isto é, o algoritmo desenvolvido encontra a solução ótima como a Formulação Ciclo mas o seu tempo de execução é bastante

superior à resolução da formulação ciclo.

Para tentar melhorar o algoritmo, foi necessário analisar detalhadamente a aplicação do algoritmo a exemplos pequenos para tentar encontrar uma condição que tornasse a pesquisa mais inteligente e consequentemente diminuisse o tempo de execução do algoritmo. Quando se realizou tal análise, verificou-se que o problema era que durante a pesquisa o algoritmo ao selecionar um ciclo para analisar, o ramo esquerdo por este criado iria possuir um majorante igual ou inferior ao valor da melhor solução atual o que fazia com que este ramo fosse cortado. Ou seja, a pesquisa em árvore estava a "gastar"tempo a analisar um ciclo que não conduzia a um melhoramento da solução atual porque possuía um valor do majorante menor ou igual ao tamanho da melhor solução. Esta situação acontecia sucessivamente fazendo com que a árvore analisasse inúmeros nós que acabariam por ser cortados, como se pode visualizar na Figura 4.16. Perante esta situação, constatou-se que o problema estava na seleção do ciclo a analisar no nó seguinte.

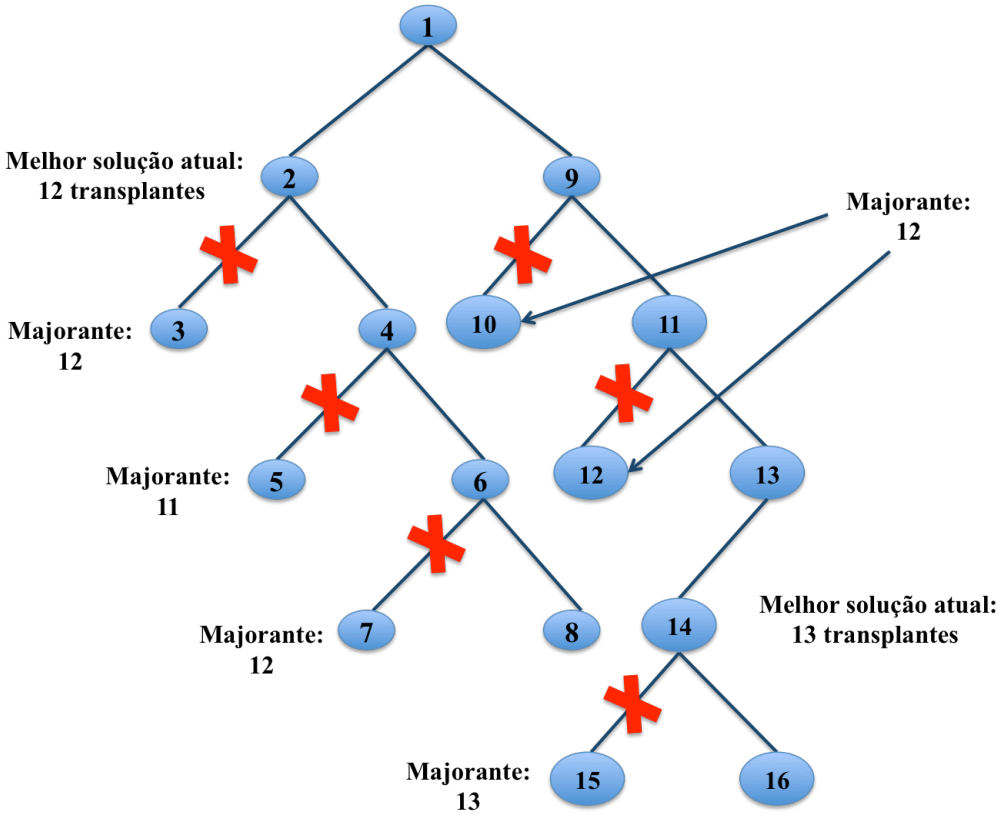


Figura 4.16: Exemplo ilustrativos de sucessivos nós cortados

Sendo assim, o estudo focou-se em determinar uma condição para selecionar o ciclo a ser analisado ou então uma condição que filtrasse os ciclos disponíveis em cada nó, de modo a diminuir o número de ciclos disponíveis (eliminando aqueles que iriam originar ramos com majorante igual ou inferior ao tamanho da melhor solução).

Regra para seleção de ciclos

Relativamente à seleção do ciclo a ser analisado, a primeira ideia que surgiu para tentar melhorar a execução do programa foi primeiramente serem selecionados os ciclos pertencentes ao digrafo constituído pelos arcos da solução do majorante. Para instâncias de teste com poucos vértices (10 ou 20), o algoritmo estava a demorar mais do que era normal, uma vez que era necessário executar a função *ciclos* de cada vez que se calculava o majorante do ramo esquerdo. Mas, mesmo assim esta condição foi testada para instâncias superiores. Nomeadamente, instâncias com 40, 50 e 60 vértices para verificar se melhorava consideravelmente o tempo de execução do algoritmo. Tal não aconteceu e concluiu-se assim que esta condição não era eficiente e portanto não foi inserida no algoritmo.

Regra para filtragem de ciclos

A segunda ideia que se tentou aplicar foi na tentativa de filtrar os ciclos disponíveis em cada nó. No caso do ramo esquerdo neste momento a obtenção dos ciclos disponíveis consistia em dado o conjunto dos ciclos disponíveis do nó pai remover aqueles que possuissem um vértice em comum com o ciclo selecionado. Para o ramo direito os ciclos disponíveis determinavam-se através da remoção do conjunto dos ciclos avaliados ao conjunto dos ciclos disponíveis do nó pai. A ideia era criar uma condição que para cada ramo, perante os ciclos disponíveis obtidos pelo processo descrito acima, filtrasse os ciclos que realmente valiam a pena avaliar.

Essa condição consistia em dada a lista dos ciclos disponíveis, para cada um deles verificar se quando este era selecionado a solução obtida era superior à melhor solução atual. Por outras palavras, para cada ciclo da lista considerava-se que esse ciclo era selecionado e removia-se do digrafo atual. De seguida construia-se o grafo correspondente e encontrava-se a solução de Edmonds para esse grafo. Se o tamanho da solução, no caso de se tratar de um ramo esquerdo, constituída pela solução de Edmonds mais os dois ciclos em questão (o ciclo que está a ser analisado nesse ramo e o que foi selecionado da lista dos ciclos disponíveis) fosse superior ao tamanho da melhor solução atual o ciclo permanecia na lista dos ciclos disponíveis. No caso do ramo direito, o tamanho da solução era dado pela solução de Edmonds mais o ciclo da lista disponível selecionado, uma vez que este ramo corresponde a não usar o ciclo em análise.

Inicialmente esta condição parecia que estava a resultar com bastante sucesso para as instâncias que estavam a ser testadas, dado que para uma instância com 50 vértices o tempo de execução passou de cerca de 10000 segundos para 40 segundos.

Quando se procedeu à execução do programa com esta condição para todas as instâncias teste, para as instâncias de tamanho 10, 20, e 30 os resultados obtidos eram bastante melhores em termos de tempo de execução e tinham ainda a vantagem de que com esta condição a solução encontrada para além de ser máxima possuía o número mínimo de ciclos de tamanho três. Esta característica revelava-se uma vantagem relativamente à Formulação Ciclo que se estava a usar. Diz-se que é uma vantagem em relação à formulação ciclo usada porque a Formulação Ciclo que se estava a usar tinha como objetivo maximizar o número de transplantes realizados, sem ter a restrição de utilizar primeiramente os ciclos de tamanho 2. Isto é, não tinha a condição de minimizar o número de ciclos de tamanho 3 usados. Mas é necessário referir que é possível alterar a Formulação Ciclo em uso de forma a que

esta também maximize o número de ciclos de tamanho 2 utilizados. Nesse caso o tempo de execução irá aumentar podendo mesmo assim não ser igual ou superior ao do algoritmo desenvolvido. Quando se procedeu à execução do algoritmo para instâncias de tamanho 40, verificou-se que para uma dada instância a solução encontrada através do algoritmo não correspondia à solução ótima, o que levou a concluir que a condição estava errada, uma vez que esta apenas se ia tornar numa heurística que funcionaria em determinados casos específicos.

Sendo assim, tentou-se analisar as causas da falha da condição anteriormente descrita. O que se verificou foi que esta condição estava a perder soluções que seriam encontradas posteriormente. Isto é, apesar de um determinado ciclo da lista de ciclos disponíveis não estar a fornecer uma solução melhor aquando da sua utilização pode posteriormente em conjunto com outros ciclos formar uma solução melhor do que a atual. Estas situações estavam a ser descartadas pela condição imposta.

Para concluir, a informação que se pretende transmitir com a análise do algoritmo é que apesar da Formulação Ciclo apresentar resultados mais eficazes do ponto de vista de CPU, não significa que a utilização de uma pesquisa em árvore não seja um bom método de resolução deste problema. Pode ser possível, através da exploração das características do algoritmo, encontrar uma condição que reestruture a pesquisa em árvore tornando-a mais inteligente e consequentemente uma diminuição do tempo de execução da pesquisa. Também é possível explorar a ideia de maximizar os ciclos de tamanho 2 utilizados, uma vez que isto se traduz em vantagens a nível médico.

Capítulo 5

Apresentação e Discussão dos Resultados Computacionais Obtidos

Neste capítulo são descritos os dados utilizados e os resultados computacionais obtidos pelo algoritmo mencionado neste trabalho. Uma vez que não existem dados reais suficientes para serem realizados testes computacionais, foi necessário recorrer a um gerador de dados para este tipo de problema. Primeiramente será descrita a forma como o gerador em questão cria os dados necessários para os testes e de seguida são apresentados os respetivos resultados computacionais juntamente com uma pequena análise e discussão destes.

5.1 Descrição dos dados

As instâncias utilizadas para testar o algoritmo criado para o problema de transplantação renal emparelhada são geradas usando um simulador de potenciais dadores e pacientes de trocas de rins (com o qual se constroem as bases de dados correspondentes à listagem dos pares dador/recetor incompatíveis que querem participar no programa de transplantação renal emparelhada). Este simulador, está escrito na linguagem C^{++} e foi criado por (Saidman et al., 2006). Os pares dador/recetor são gerados com um tipo de sangue aleatório tanto para o dador como para o recetor, um tipo de sexo aleatório para recetor e uma relação aleatória (familiar, amigo etc) entre o recetor e o dador. Ao recetor de cada par também é atribuída uma probabilidade aleatória para a incompatibilidade com o tipo de tecido do dador. Se o recetor e o dador correspondente forem incompatíveis, são inseridos na base de dados.

Em seguida o gerador verifica o tipo de sangue e a incompatibilidade do tipo de tecido de cada recetor j com todos os dadores, exceto o seu dador correspondente. Se o dador i for compatível com o recetor j, isto é, se for possível efetuar um transplante entre o dador i e recetor j, o gerador cria o arco (i,j) (que significa que vai do vértice i para o vértice j) no grafo correspondente. O peso de cada vértice foi definido como 1, porque se presume que a probabilidade de sobrevivência de um rim após um transplante não depende do dador escolhido, desde que o dador e o recetor sejam compatíveis (Delmonico, 2004). Depois de verificar todos os recetores, o gerador produz todos os vértices e arcos do grafo.

De seguida, apresenta-se a Figura 5.1 onde se pode ter uma visão esquemática dos programas e das dependências existentes.

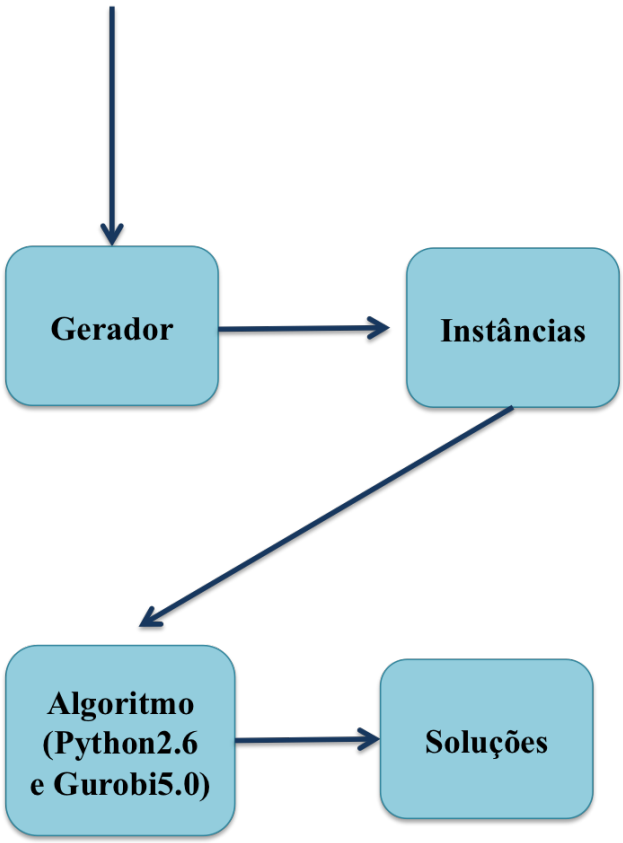


Figura 5.1: Visão Esquemática dos programas e dependências

5.2 Resultados Computacionais

Os resultados computacionais foram obtidos utilizando um computador MacBook Pro com processador Intel i5 a 2.3Ghz, 4Gb de memória e executados no sistema operacional Mac OS X 10.7.4; foram também utilizados os softwares *Python2.6* (van Rossum, 1995) e Gurobi *5.0* (Gurobi Optimization, Inc., 2012). É importante referir que, na execução de algumas das instâncias o computador sofreu um sobreaquecimento que pode ter influenciado o desempenho do CPU nas execuções posteriores (aumentando o tempo de execução do algoritmo).

Para comparar o algoritmo desenvolvido com a formulação ciclo para o problema de transplantação renal emparelhada, foram geradas 50 instâncias com um número de vértices (número de pares dador/recetor) 10, 20, 30, 40, 50 e 60. Para cada uma das instâncias foi testado o algoritmo com pesquisa em largura e com pesquisa em profundidade, e aplicada a formulação ciclo, a fim de depois ser comparado o tempo de execução de cada um dos

métodos e analisar algumas características existentes nos resultados obtidos.

Dado que, como referido na Secção 4.2.3, o tempo de execução do algoritmo desenvolvido por vezes é bastante superior ao tempo de execução da Formulação Ciclo, foi criada uma condição através da qual todas as instâncias cujo tempo de execução de uma das pesquisas é superior a 3600s não são utilizadas para efetuar a análise do algoritmo. Isto é, nos resultados apresentados nesta secção não existe informação referente às instâncias cujo tempo de execução da aplicação de uma das versões do algoritmo foi superior a 3600s.

Por consequência da condição imposta, o número de instâncias de tamanho 40 que serão utilizadas para analisar o desempenho do algoritmo para este tamanho serão apenas 30, porque as restante 20 instâncias possuem tempo de execução superior a 3600s. No caso dos tamanhos 50 e 60 não serão apresentados quaisquer resultados numéricos nem gráficos porque, após a aplicação do algoritmo às instâncias correspondentes a estes tamanhos, verificou-se que o número de instâncias com tempo de execução inferior a 3600s era práticamente nulo. Sendo assim, para efeitos de análise e discussão dos resultados apenas será tida em conta a informação de que o tempo de execução do algoritmo desenvolvido para instâncias de tamanho 50 e 60 é quase sempre superior a 3600s.

A apresentação dos resultados computacionais vai ser dividida por cada tamanho considerado, ou seja, primeiramente serão apresentados os resultados referentes às 50 instâncias de tamanho 10, depois os resultados referentes às 50 instâncias de tamanho 20 e assim sucessivamente até às instâncias de tamanho 40. Para cada um dos tamanhos apresentados dado o elevado número de instâncias geradas, nesta secção apenas será apresentada uma tabela resumo onde é representada a média do tempo de execução de cada método aplicado. No entanto, em apêndice será colocada uma tabela com todos os detalhes da aplicação de cada método, nomeadamente, o tempo de execução, o número de ciclos existentes inicialmente no digrafo e o número de nós visitados na pesquisa em árvore. Serão também apresentados três gráficos, onde num são desenhadas as curvas referentes ao tempo de execução de cada método (uma curva para o algoritmo de pesquisa em largura, outra para o algoritmo de pesquisa em profundidade e outra para a formulação ciclo) a fim de posteriormente estes serem comparados entre si. Apresenta-se ainda outro gráfico onde é representado o tempo de execução em função do número de ciclos de tamanho 3 para cada uma das instâncias, para que se possa perceber se existe alguma relação entre esses valores. E, por último, um outro gráfico onde para cada instância é apresentado a média do número de ciclos com vértices em comum. Através deste gráfico pretende-se analisar a eficiência da condição apresentada no **Capítulo 6** como trabalho futuro.

A análise e discussão dos resultados obtidos será efetuada após a apresentação de todos os resultados.

Resultados referentes às instâncias de tamanho 10

Na tabela 5.1 estão representadas as médias do tempo de execução para cada um dos métodos aplicados e na Figura 5.2 o respetivo gráfico de pontos. Na Figura 5.3 é representada a relação entre o número de ciclos de tamanho três existentes numa instância e o tempo de execução necessário para cada método. Em ambos os gráficos a escala utilizada no eixo das ordenadas é logarítmica. Por último a Figura 5.4, apresenta a média do número de ciclos com vértices em comum. Para a elaboração deste gráfico foi utilizada a escala linear. A totalidade dos resultados correspondente a estas instâncias encontra-se no **Apêndice B**.

Método	**Média tempo de execução**
Formulação Ciclo	0.0397
Pesquisa em Profundidade	0.0515
Pesquisa em Largura	0.0102

Tabela 5.1: Média do tempo de execução para as instâncias de tamanho 10

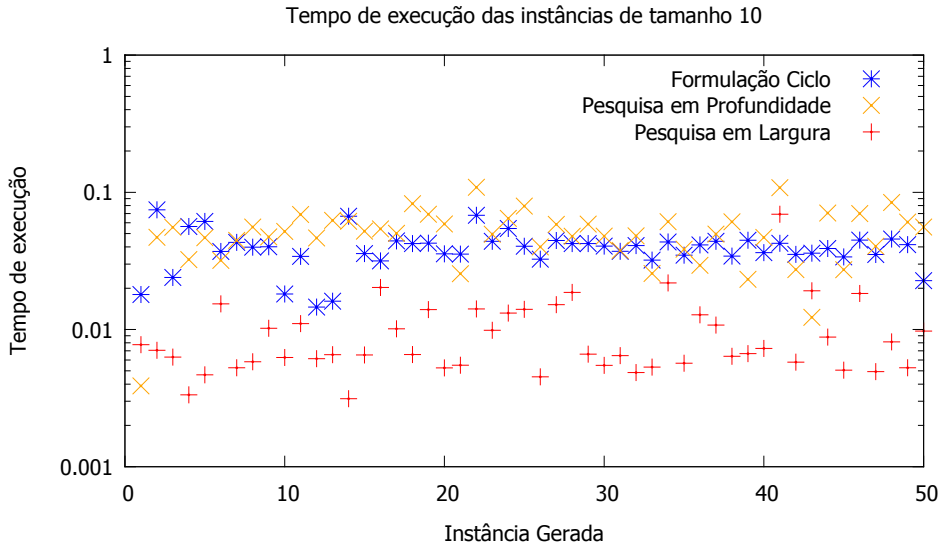


Figura 5.2: Tempo de Execução para as instâncias de tamanho 10

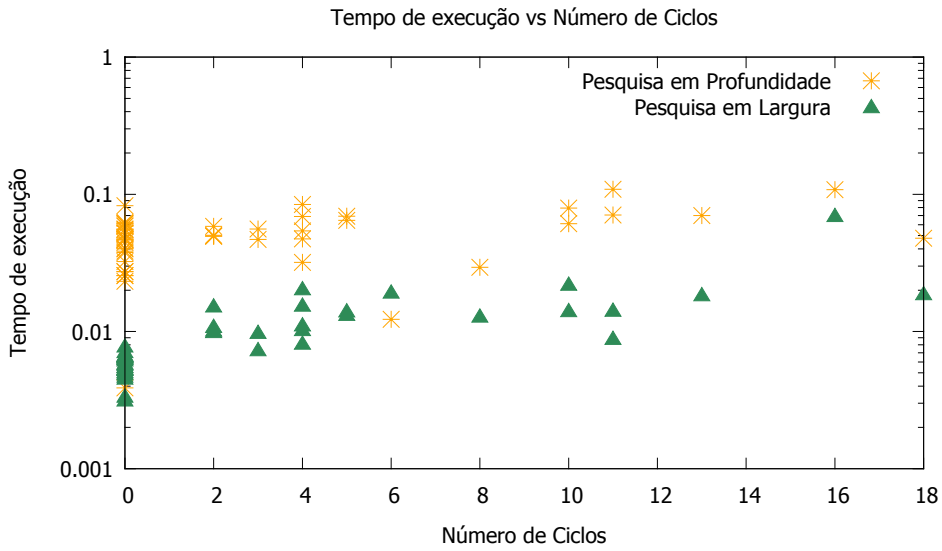


Figura 5.3: Tempo de Execução vs Número de Ciclos para as instâncias de tamanho 10

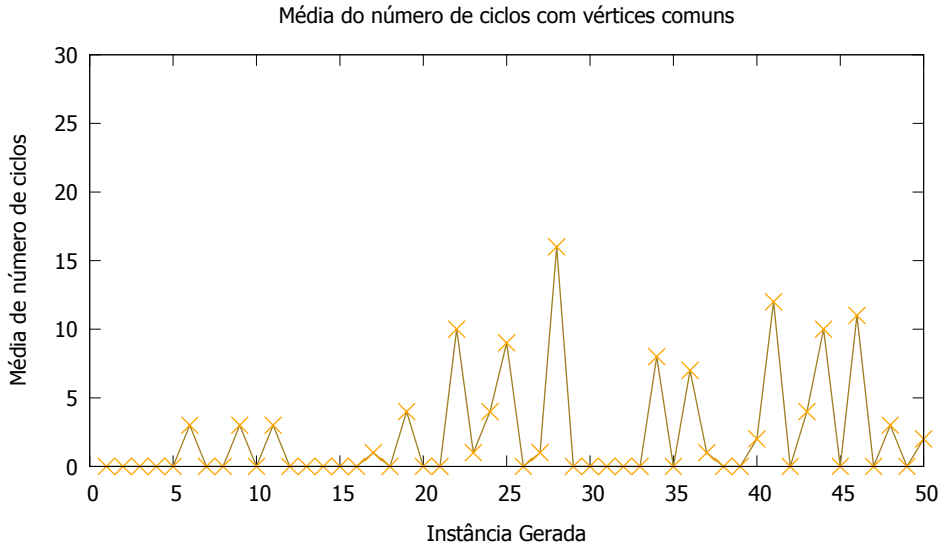


Figura 5.4: Média do número de ciclos com vértices em comum para as instâncias de tamanho 10

Resultados referentes às instâncias de tamanho 20

A tabela 5.2 corresponde à média do tempo de execução de cada método aplicado ao conjunto de instâncias de tamanho 20. Nesta tabela está também representada a média do tempo

de execução obtida considerando a exclusão da instância que em comparação com todas as outras possui um tempo de execução do algoritmo desenvolvido drasticamente superior (instância nº15). As razões pelas quais esta instância apresenta um tempo de execução tão superior são o elevado número de ciclos de tamanho 3 que esta possui (esta instância tem 145 ciclos de tamanho 3) e o facto de a média de ciclos com vértices em comum não ser muito próxima do número de ciclos (a média do número de ciclos de vértices em comum é 96). Isto traduz-se na criação de inúmeros ramos esquerdos que posteriormente acabam por ser cortados devido ao valor do majorante que possuem. Toda a informação obtida relativamente a este conjunto de instâncias pode ser visualizada no **Apêndice C**.

Método	**Média tempo de execução**	**Média tempo de execução sem a instância crítica**
Formulação Ciclo	0.0522	0.0509
Pesquisa em Profundidade	2.3958	0.3672
Pesquisa em Largura	2.1964	0.3272

Tabela 5.2: Média do tempo de execução para as instâncias de tamanho 20

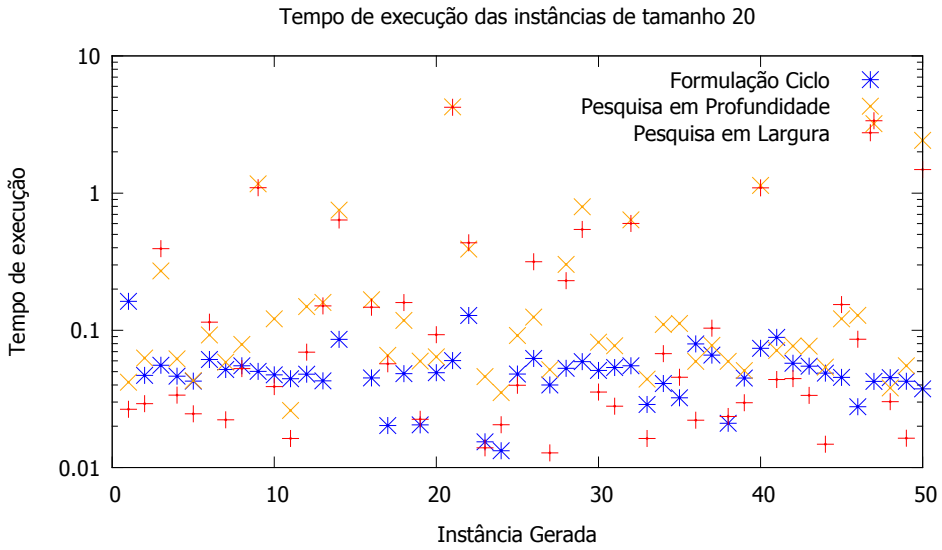


Figura 5.5: Tempo de Execução para as instâncias de tamanho 20

A existência de uma instância que em comparação com as restantes possui um tempo de execução drasticamente superior, fez com que a utilização da escala linear no eixo das ordenadas impossibilitasse uma análise adequada do gráfico de pontos do tempo de execução de cada método. Isto porque, como existe uma elevada discrepância entre o tempo de execução máximo e os restantes valores, o desenho dos pontos do tempo de execução sobrepõe-se praticamente ao eixo das abcissas com exceção no já referido tempo de execução máximo. Posto isto, tanto o gráfico de pontos (Figura 5.5) como o gráfico Tempo de Execução vs Número de Ciclos (Figura 5.6) foram construídos utilizando a escala logarítmica. É ainda importante ressalvar que nestes gráficos é possível visualizar-se toda

a informação referente ao comportamento do tempo de execução dos métodos aplicados, incluindo o tempo de execução drasticamente superior. O gráfico da Figura 5.7 possui escala linear e representa para cada instância a média do número de ciclos com vértices em comum.

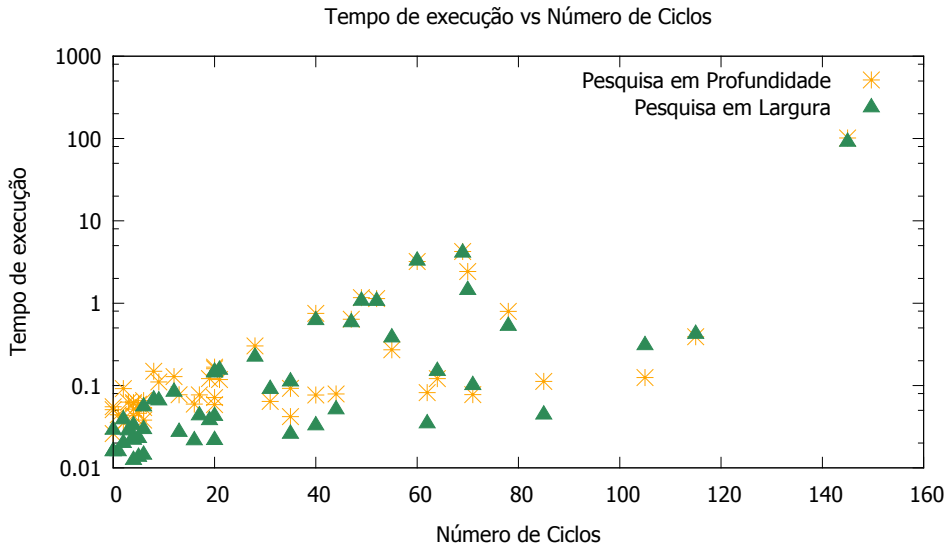


Figura 5.6: Tempo de Execução vs Número de Ciclos para as instâncias de tamanho 20

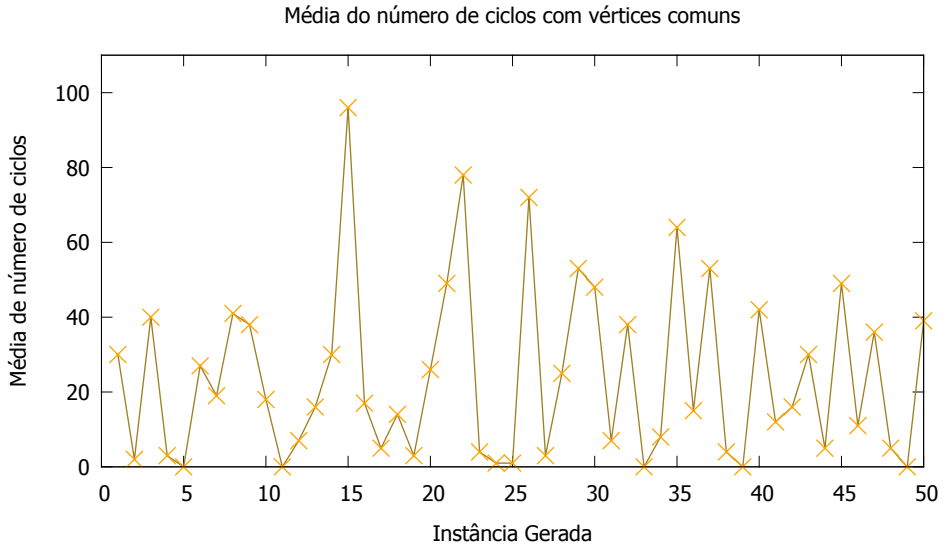


Figura 5.7: Média do número de ciclos com vértices em comum para as instâncias de tamanho 20

Resultados referentes às instâncias de tamanho 30

Na tabela 5.3 são apresentadas as médias do tempo de execução para cada um dos métodos

aplicados. Após a análise dos tempos de execução obtidos para os dois tipos de pesquisa verificou-se que existe apenas uma instância para a qual a pesquisa em largura apresenta um tempo de execução consideravelmente superior à pesquisa em profundidade (instância nº24). Sendo assim, na tabela 5.3 é também apresentada a média do tempo de execução de cada método obtida após ter sido retirada a instância crítica. É importante referir as razões que fazem com que esta instância seja crítica. O que acontece na pesquisa em largura para esta instância é o problema do algoritmo desenvolvido referido anteriormente. Isto é, devido à forma como a pesquisa se processa o número de ramos criados que acabam por ser cortados aumentou drásticamente, atrasando a determinação da solução ótima. Por exemplo, enquanto que na pesquisa em profundidade o 1º ciclo de tamanho 3 que melhora a solução foi encontrado no segundo nó visitado, na pesquisa em largura este aprece apenas no sexto nó visitado. Para este conjunto de instâncias o **Apêndice D** contém toda a informação obtida.

Método	**Média tempo de execução**	**Média tempo de execução sem a instância crítica**
Formulação Ciclo	0.0655	0.0634
Pesquisa em Profundidade	4.9635	4.7978
Pesquisa em Largura	6.3375	4.7998

Tabela 5.3: Média do tempo de execução para as instâncias de tamanho 30

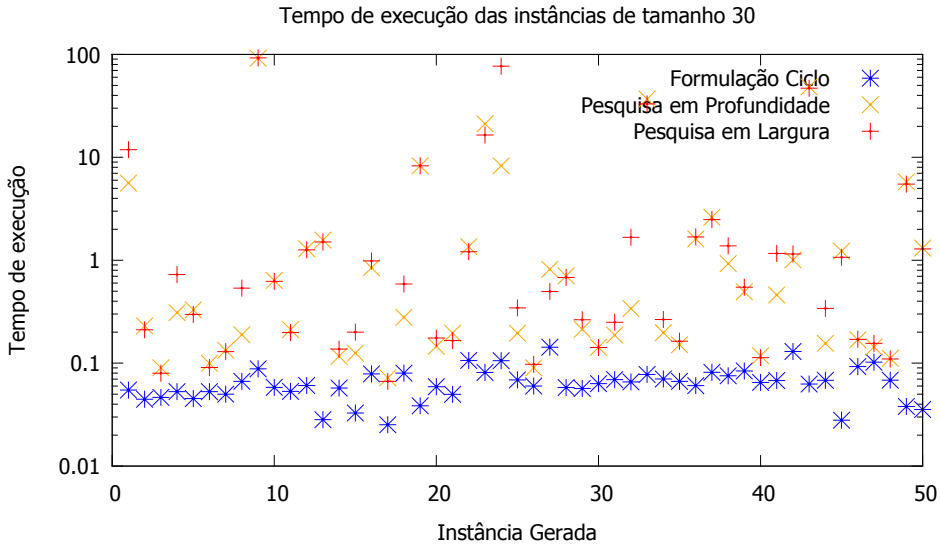


Figura 5.8: Tempo de Execução para as instâncias de tamanho 30

Nas Figuras 5.8 e 5.9 estão representados o gráfico de pontos do tempo de execução de cada um dos métodos e, a relação entre o número de ciclos de tamanho 3 existentes numa instância e o tempo de execução que esta obteve em cada método, respetivamente. Em ambos os gráficos é utilizada a escala logarítmica no eixo das ordenadas e é possível observar-se a

instância crítica descrita anteriormente. A Figura 5.10 representa a média do número de ciclos de tamanho 3 com vértices em comum para cada instância, e utiliza escala linear.

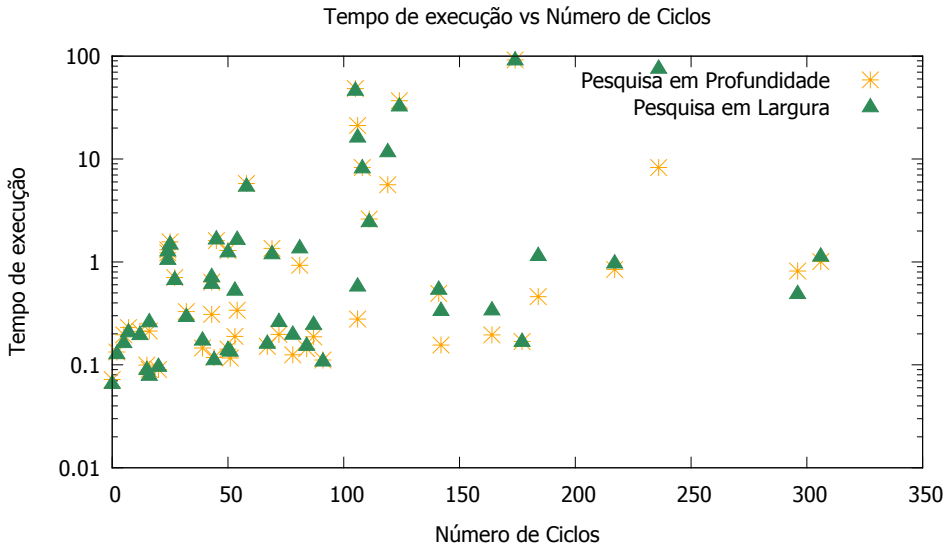


Figura 5.9: Tempo de Execução vs Número de Ciclos para as instâncias de tamanho 30

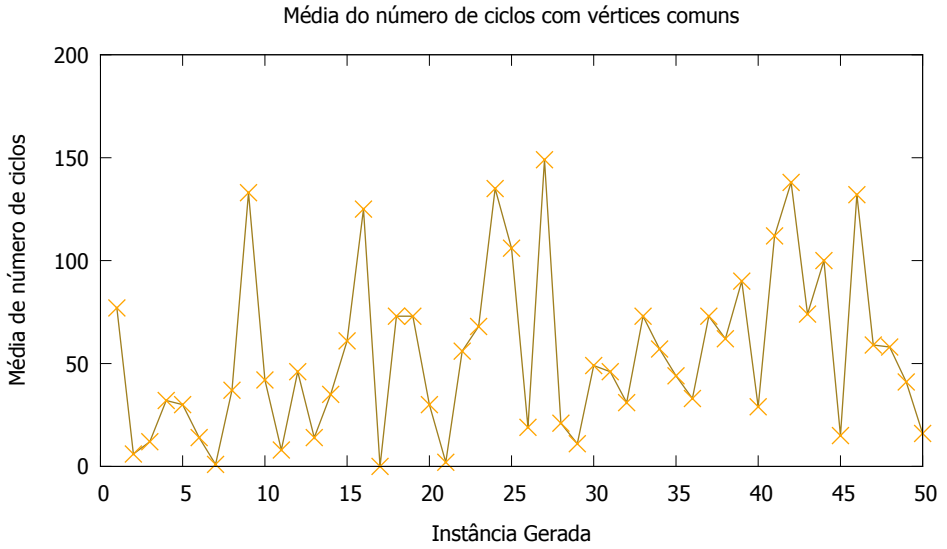


Figura 5.10: Média do número de ciclos com vértices em comum para as instâncias de tamanho 30

Resultados referentes às instâncias de tamanho 40

Para o conjunto de instâncias de tamanho 40 a média do tempo de execução de cada um

dos métodos encontra-se representada na tabela 5.4. O gráfico de pontos do tempo de execução está representado na Figura 5.11, e a relação entre o tempo de execução e o número de ciclos de tamanho três existentes em cada instância está representada na Figura 5.12. Para a construção destes gráficos é uma vez mais utilizada a escala logarítmica no eixo das ordenadas. Na Figura 5.13 está representada a média do número de ciclos com vértices em comum em cada instância. Para este último gráfico foi utilizada a escala linear. O **Apêndice E** possui toda a informação detalhada associada a este conjunto de instâncias.

Método	**Média tempo de execução**
Formulação Ciclo	0.108
Pesquisa em Profundidade	255.4365
Pesquisa em Largura	285.1469

Tabela 5.4: Média do tempo de execução para as instâncias de tamanho 40

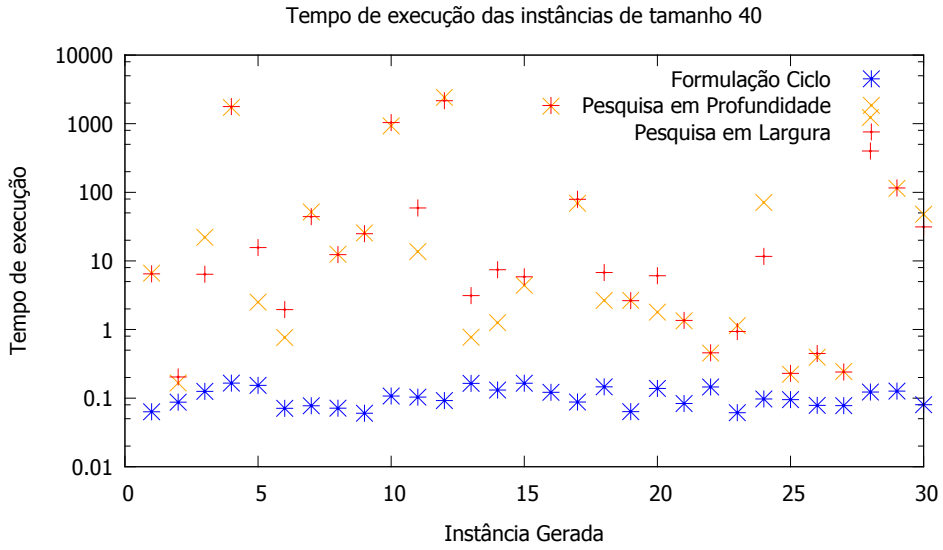


Figura 5.11: Tempo de Execução para as instâncias de tamanho 40

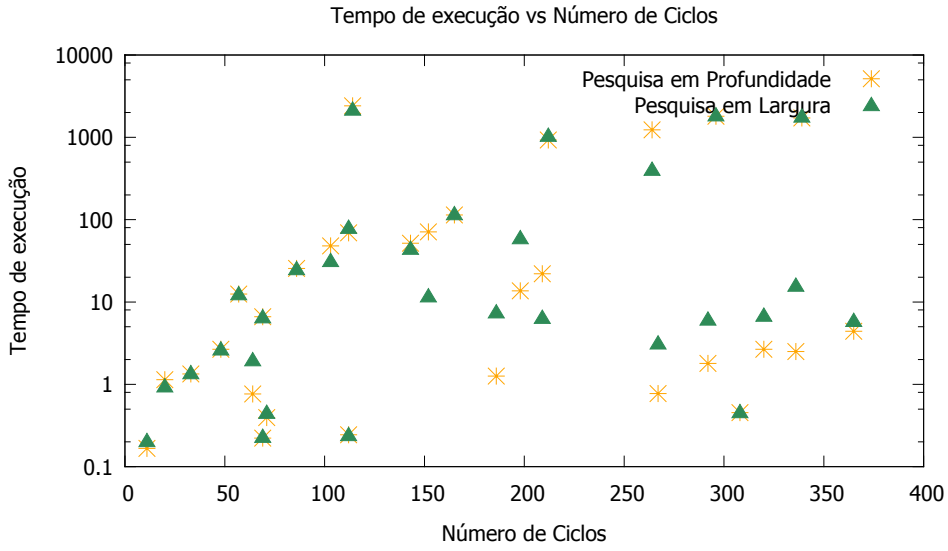


Figura 5.12: Tempo de Execução vs Número de Ciclos para as instâncias de tamanho 40

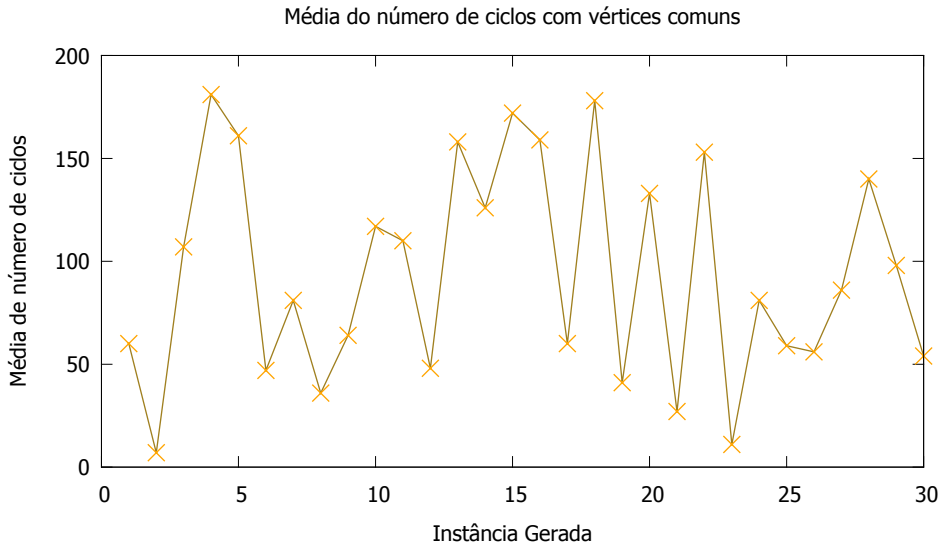


Figura 5.13: Média do número de ciclos com vértices em comum para as instâncias de tamanho 40

5.3 Análise dos Resultados

Após a apresentação de toda a informação obtida da aplicação do algoritmo desenvolvido, efetua-se a análise dos resultados.

Inicialmente será analisada a evolução do tempo de execução ao longo da aplicação do algoritmo às diferentes instâncias geradas. No que diz respeito às instâncias de tamanho 10, a média do tempo de execução obtida é bastante satisfatória, sendo que a média do tempo de execução do algoritmo com pesquisa em largura possui valor inferior à média do tempo de execução da formulação ciclo. Pode-se constatar esta informação no gráfico 5.2, uma vez que a curva correspondente ao tempo de execução do algoritmo com pesquisa em largura está praticamente sempre abaixo da curva que representa o tempo de execução para a formulação ciclo.

No caso das instâncias de tamanho 20, a média do tempo de execução para as duas pesquisas é superior à Formulação Ciclo, existindo mesmo uma instância para a qual se verifica uma diferença muito elevada entre os 3 tempos de execução correspondentes (Figura 5.5). É importante referir que, apesar da média do tempo de execução para as duas pesquisas ser superior à média do tempo de execução da Formulação Ciclo, a diferença entre estas não é muito superior como se pode constatar na Tabela 5.2. Em particular, no caso em que a instância crítica (instância com tempo de execução drasticamente superior) é removida, a média do tempo de execução das duas pesquisas diminui consideravelmente, aproximando-se da média da Formulação Ciclo.

O comportamento da média do tempo de execução da pesquisa em árvore para as duas variantes, no caso das instâncias de tamanho 30, é análogo ao comportamento para as instâncias de tamanho 20, sendo necessário realçar apenas a diferença considerável entre os tempos médios de execução do Algoritmo com Pesquisa em Largura e com Pesquisa em Profundidade (Tabela 5.3). É ainda importante salientar que com a remoção da instância crítica (com grande discrepância entre os tempos de execução das duas pesquisas), a média do tempo de execução da Pesquisa em Largura diminui significativamente, tornando-se idêntica à média da Pesquisa em Profundidade.

Por último, para as instâncias de tamanho 40, verificou-se uma drástica subida do tempo médio de execução do algoritmo desenvolvido, quer utilizando pesquisa em largura, quer utilizando pesquisa em profundidade (Tabela 5.4). No gráfico 5.11 é possível verificar-se esta discrepância. Isto porque, a curva correspondente ao tempo de resolução da formulação ciclo encontra-se significativamente abaixo das outras curvas, o que mostra que esta possui valores muito inferiores comparativamente com os valores das outras curvas (tempo de execução do algoritmo com pesquisa e largura e tempo de execução do algoritmo com pesquisa em profundidade).

Adicionando a toda esta informação o facto de que para as instâncias de tamanho 50 e 60 o Algoritmo desenvolvido possuir quase sempre tempo de execução superior a 3600s (para os dois tipos de pesquisa), pode-se concluir que o tempo de execução do algoritmo para problemas de tamanho reduzido (10, 20, e até mesmo 30) apresenta um comportamento razoável, com uma média de tempo de execução não muito superior à Formulação Ciclo. No

caso de problemas de dimensão superior, o tempo de execução aumenta drasticamente com o aumento do tamanho das instâncias e o algoritmo deixa de ser competitivo.

Para tentar obter mais informação característica do desempenho da pesquisa em árvore desenvolvida, criou-se um gráfico onde para cada instância está representada a relação entre o tempo de execução e o número de ciclos de tamanho três existentes nessa instância. A fim de tentar verificar se as instâncias com maior tempo de execução coincidem com as que possuem maior número de ciclos de tamanho três.

Analisando cada um dos gráficos correspondentes aos diferentes tamanhos de instâncias considerados verifica-se que, por exemplo, no caso das instâncias de tamanho 10 o maior tempo de execução está associado a uma instância com 3 ciclos de tamanho três e existem instâncias de tamanho 10 com maior número de ciclos de tamanho três. Isto é, para este conjunto de instâncias não se verifica a relação de que o maior tempo de execução está associado à instância com maior número de ciclos de tamanho três. Apenas no caso das instâncias de tamanho 20 é que se verifica esta relação, como se pode constatar no gráfico 5.6. Para as restantes instâncias de tamanho superior o maior tempo de execução não está associado ao maior número de ciclos de tamanho 3 (Gráficos 5.9 e 5.12).

Relativamente aos gráficos 5.4, 5.7, 5.10 e 5.13 o que se pode concluir é que todas as instâncias teste possuem um número considerável de ciclos com vértices em comum. Existem mesmo casos em que todos os ciclos de tamanho 3 existentes na instância teste possuem vértices em comum. Esta observação reforça a eficácia da condição pensada apenas teóricamente, proposta como trabalho futuro. Isto porque, se existe um elevado número de ciclos com vértices em comum, ao analisar-se a disjunção entre estes, muitos ciclos são excluídos. Ou seja, a utilização da condição sugerida no **Capítulo 5** faz com que o número de ciclos disponíveis em cada nó diminua significativamente e consequentemente o tempo de execução da pesquisa em árvore diminua.

A análise final aos resultados obtidos neste trabalho traduz que o algoritmo desenvolvido em relação ao tempo de execução, apresenta resultados relativamente piores comparativamente com a formulação ciclo. Verificou-se também que não é possível retirar nenhuma relação entre o tempo de execução e o número de ciclos de tamanho 3 que uma instância contém. Mais uma vez, é importante ressalvar que, apesar dos resultados computacionais obtidos não se conclui que o desenvolvimento de uma pesquisa em árvore para a resolução do problema em estudo não seja um bom método de resolução. Isto porque é possível explorar a estrutura da pesquisa em árvore criada de forma a encontrar uma nova condição que melhore os resultados. Ou então, uma análise ainda mais detalhada do algoritmo de Edmonds pode revelar uma característica do algoritmo que se traduza numa condição que melhore o desempenho da pesquisa em árvore.

Capítulo 6

Conclusão e Trabalhos Futuros

Este trabalho consistiu no estudo do problema Transplantação Renal Emparelhada e inclui, entre outros, a definição e desenvolvimento de uma base de dados associada a um algoritmo de otimização para a resolução do problema pela ASST, o estudo e compreensão de um algoritmo complexo (Algoritmo de Edmonds) e desenho e implementação de um algoritmo para a resolução do problema baseado numa pesquisa em árvore.

As primeiras conclusões que se puderam retirar com a elaboração deste projeto são referentes à base de dados criada inicialmente e a toda a sua utilidade. Anteriormente a filtragem dos pares dador/recetor com compatibilidade sanguínea e imunológica era efetuada manualmente pela ASST. Com a criação da base de dados todo esse processo tornou-se automático através de um interface elaborado em Excel juntamente com a aplicação VBA. Este facto permite realçar uma das vantagens associadas à base de dados, uma vez que com a automatização das filtragens a ASSST deixou de despender um elevado período de tempo para determinar os pares compatíveis. Esta vantagem torna-se ainda mais valiosa com o aumento do número de pares dador/recetor inscritos no PNDRC.

O outro grande benefício associado à criação da base de dados é que esta acopla um modelo de otimização através do qual, dados os pares dador/recetor compatíveis, se consegue determinar o número máximo de transplantes que se podem realizar e os respetivos emparelhamentos. Ou seja, permite obter a solução do problema em estudo para os pares dador/recetor registados no PNDRC em Portugal. Esta ferramenta (base de dados e modelo de otimização) permitiu ao longo deste projeto, em cada reunião realizada na ASST, verificar se mediante os pares dador/recetor inscritos no programa existia algum transplante renal que se pudesse efetuar.

O Problema da Transplantação Renal Emparelhada para instâncias de tamanho pequeno/médio pode ser resolvido de forma rápida através de resolutores (solvers) de Programação Inteira Mista comerciais. Os resolutores de Programação Inteira Mista gratuitos são muito menos eficientes. Este facto pôde ser constatado através da aplicação do software GLPK a algumas instâncias teste deste projeto. Sendo assim, o que se pretendia desenvolver com a elaboração deste trabalho era um algoritmo independente de um software comercial e que se revelasse mais eficiente relativamente à Formulação Ciclo implementada no software Gurobi.

Relativamente à eficiência do algoritmo desenvolvido verificou-se que este comparativamente com a Formulação Ciclo apresenta tempo de CPU superior, isto é, torna-se menos eficiente que a Formulação Ciclo. Mas, não significa que o desenvolvimento de uma Pesquisa em Árvore para a resolução deste problema tenha sido uma má escolha, porque dada a complexidade do Algoritmo de Edmonds é possível que através de uma análise ainda mais detalhada deste, se encontre uma condição que torne a pesquisa muito mais eficiente, inteligente e consequentemente com tempo de CPU mais reduzido.

Em relação à independência do algoritmo desenvolvido relativamente a softwares comerciais, embora os testes apresentados na secção 4.2 tenham sido obtidos recorrendo ao software Gurobi, a estrutura deste algoritmo pode ser programada sem ser necessário utilizar um software comercial. Esta possibilidade foi mesmo testada através de pacotes existentes no python que possuem o Algoritmo Húngaro implementado, mas o tempo de execução da aplicação destes pacotes era ligeiramente superior ao tempo de execução da formulação implementada no Gurobi, pelo que se decidiu optar por utilizar o software Gurobi.

Uma outra vantagem que pode ser descoberta neste algoritmo, é a adaptação deste procedimento ao caso em que se considera a inclusão de dadores altruístas na base de dados. Isto porque, em estudos efetuados noutros programas, nomeadamente no do Reino Unido, ao serem realizados diversos testes nestas condições verificou-se que a Formulação Ciclo falhava. Desde então têm sido estudados métodos de decomposição, como por exemplo o método de geração de colunas com o objetivo de encontrar uma aborgadem eficiente para esta variante do problema. Deste modo é possível tentar definir e implementar heurísticas no algoritmo desenvolvido de forma a que este seja eficaz para o caso em que são considerados dadores altruístas.

Relativamente ao algoritmo é ainda importante referir como trabalho futuro, uma condição que apenas foi pensada teóricamente. Esta condição tem como objetivo melhorar o desempenho da pesquisa em árvore e consiste em dado o conjunto dos ciclos disponíveis, para cada um destes construir uma lista com os conjuntos disjuntos de ciclos que é possível formar. Dados esses conjuntos disjuntos verificar se algum deles proporciona uma solução melhor do que a melhor solução atual. Se tal acontecer significa que vale a pena analisar esse ciclo. De forma mais prática, fixando um ciclo c se existir nos ciclos disponíveis uma lista constituída por ciclos disjuntos (que contenha c) cujo emparelhamento obtido seja superior à melhor solução atual significa que, através do ciclo c é possível encontrar, num dado momento da pesquisa, uma solução melhor do que a atual e portanto c permanece na lista dos ciclos disponíveis.

Por último é necessário referir que a elaboração deste algoritmo deixou um campo aberto para desenvolvimento futuro explorando, por exemplo, a determinação de uma condição que torne a pesquisa mais eficiente para o problema estudado ou a aplicação deste método a outras variantes do problema.

Referências

Abraham, D. J. and Blum, A. e. a. (2007). Clearing algorithms for barter exchange markets: Enabling nationwide kidney exchanges. In *Proceedings of the 8th ACM conference on Electronic commerce*.

Associação dos Doentes Renais do Norte de Portugal (2011). Olhe para os rins com o coração. http://www.spnefro.pt/noticias/PDFs/DMR_2011_evento_III.pdf.

ASST (2011). Programa nacional de doação renal cruzada.

Buyens, J. (2003). *Aprender mais Programação*. McGraw-Hill de Portugal, $L.^{da}$.

Delmonico, F. L. (2004). Exchanging kidneys - advances in living-donor transplantation. *New England Journal of Medicine*, 350.

Garfinkel, R. S. and Nemhauser, G. L. (1972). *Integer Programming*. John Wiley & Sons, Inc.

Gentry, S. (2008). *Optimization over graphs for kidney paired donation*, chapter Optimization in Medicine and Biology. editor.

Gurobi Optimization, Inc. (2012). Gurobi optimizer reference manual, version 5.0. http://www.gurobi.com.

IBM ILOG, Inc (2012). IBM ILOG CPLEX: High-performance software for mathematical programming and optimization. http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/.

Jungnickel, D. (2008). *Graphs, Networks and Algorithms*, volume 5. Springer, 3 edition.

National Kidney Fundation (2012). Programs for donor/recipient pairs with incompatible blood types. http://www.kidney.org/transplantation/livingdonors/incompatiblebloodtype.cfm.

Nemhauser, G. L. and Wolsey, L. (1999). *Integer and Combinatorial Optimization*. John Wiley & Sons, Inc.

Nina, N. (1999). *Visual Basic 6, Curso Completo*. FCA- Editora de Informática Lda, 3 edition.

Papadimitriou, C. and Kenneth, S. (1998). *Combinatorial Optimization Algorithms and Complexity*. Dover Publications, Inc.

portaria nº 810/2010 (2010). Programa nacional de doação renal cruzada. Diário da República.

Roth, A. E., Sönmez, T., and Utku Ünver, M. (2004). Kidney exchange. *The quarterly journal of economics*, 119.

Saidman, S., Roth, A., Sönmez, T., Unver, M., and Delmonico, F. (2006). Increasing the opportunity of live kidney donation by matching for two- and three-way exchanges. *Transplantation*.

van Rossum, G. (1995). Python reference manual.

Anexos

Anexo A

Exemplos de Pesquisa em Árvore

A.1 Exemplo de Aplicação de Pesquisa em Largura

Para se tornar mais clara a forma como se desenvolve uma Pesquisa em Largura, de seguida é apresentado na Figura A.1 um grafo e as várias etapas efetuadas ao aplicar-se este tipo de pesquisa no grafo em questão. O grafo é o seguinte:

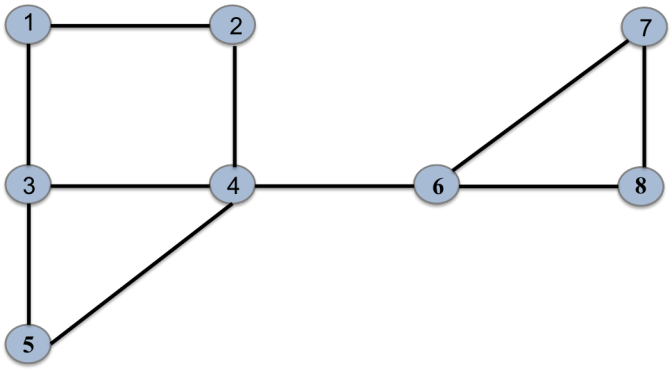


Figura A.1: Grafo Inicial para Pesquisa em Largura

Considerando que se começa a pesquisa no vértice 1, a partir deste vértice exploram-se os nós 2 e 3, pois são os únicos vizinhos que o nó 1 possui. Na figura A.2, as arestas assinaladas a cor vermelha representam os nós que já foram visitados, neste caso 2 e 3.

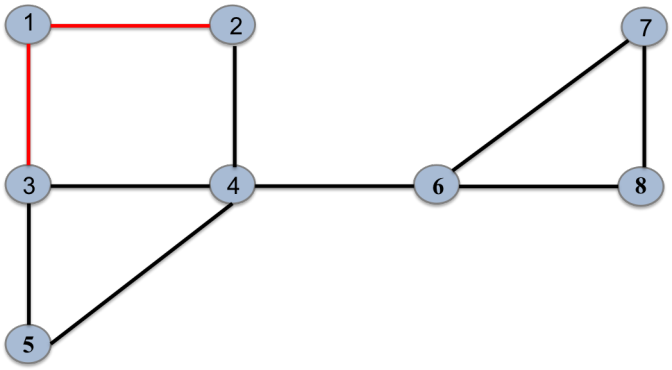


Figura A.2: Nós visitados na 1ª iteração da Pesquisa em Largura

Nesta situação pode-se prosseguir a pesquisa através do nó 2 ou do nó 3. Neste exemplo optou-se por continuar a pesquisar através do nó 3. Sendo assim, através do nó 3 visitam-se os nós 4 e 5 como se pode visualizar na Figura A.3.

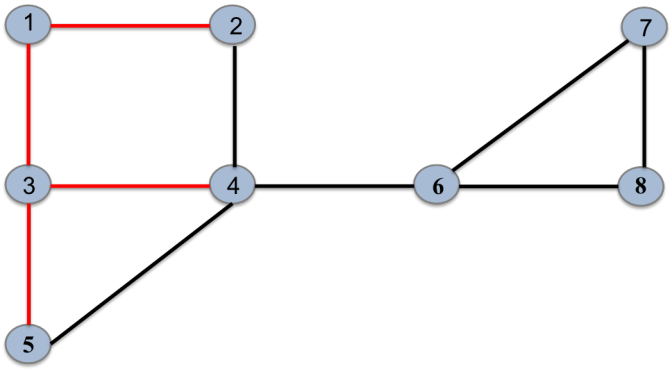


Figura A.3: Nós visitados na 2ª iteração da Pesquisa em Largura

A próxima etapa tem obrigatóriamente que ser efetuada através do nó 4, e a partir deste visita-se o nó 6. Por último, a partir do nó 6 exploram-se os nós 7 e 8, o que faz com que não existam mais nós no grafo que estejam por explorar. Sendo assim, a Pesquisa em Largura termina e o resultado final está representado na Figura A.4.

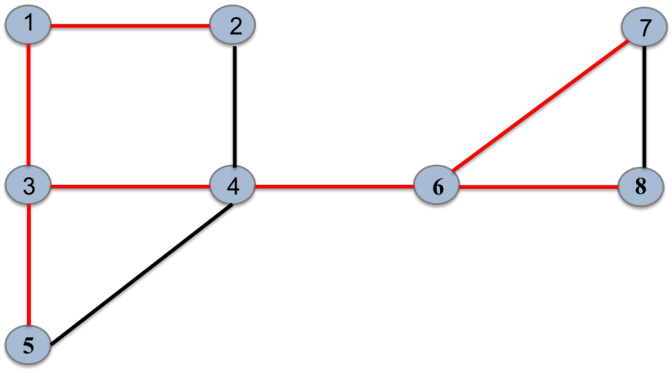


Figura A.4: Resultado final da Pesquisa em Largura

A.2 Exemplo de Aplicação de Pesquisa em Profundidade

Com o intuito de clarificar o procedimento de Pesquisa em Profundidade e até mesmo evidenciar as diferenças entre os dois tipos de pesquisa apresentados, de seguida descreve-se detalhadamente a aplicação do algoritmo de Pesquisa em Profundidade ao exemplo fornecido na Pesquisa em Largura.

Tendo em conta o grafo da figura A.1 e considerando-se que se começa a pesquisa no vértice 1, a partir deste vértice explora-se o nó 3 (podia-se ter optado por explorar o nó 2), do nó 3 segue-se para o nó 4 (mais uma vez existia outra alternativa que que era o nó 5) e por último visita-se o nó 5. Para-se no nó 5 porque este não possui nenhum vizinho para explorar. Todo este sequenciamento pode ser analisado na figura A.5, onde as arestas visitadas estão representadas a cor vermelha.

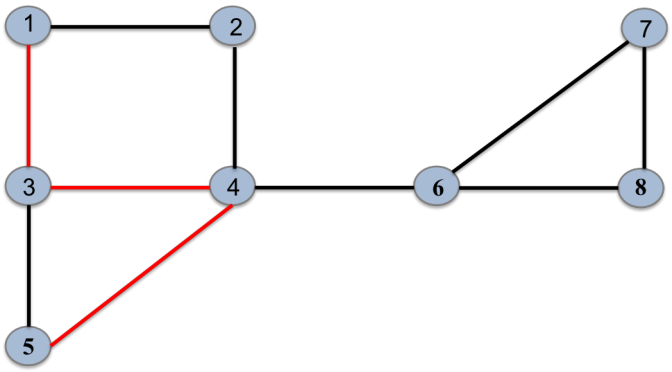


Figura A.5: Nós visitados na 1ª iteração da Pesquisa em Profundidade

Uma vez que o nó 5 não possui nenhum nó para visitar, recua-se até ao nó 4 e através deste explora-se o nó 2. É fácil notar que, não vale a pena tentar prosseguir a pesquisa através do nó 2, dado que todos os seus vizinhos já foram explorados por outros nós. Portanto, recua-se novamente ao nó 4 e visita-se o nó 6, como se pode constatar na figura A.6.

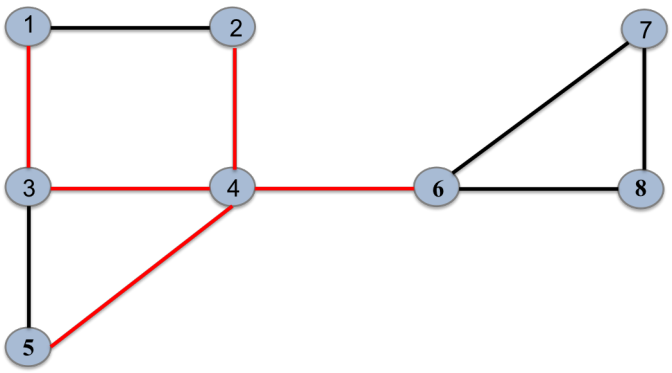


Figura A.6: Nós visitados na 2ª iteração da Pesquisa em Profundidade

Por último, do nó 6 é possível visitar o nó 7, e através deste o nó 8. Deste modo, todos os nós do grafo já foram visitados e portanto o algoritmo de Pesquisa em Profundidade termina. O esquema que representa o conjunto de todas as arestas usadas para explorar os nós do grafo está representado na Figura A.7:

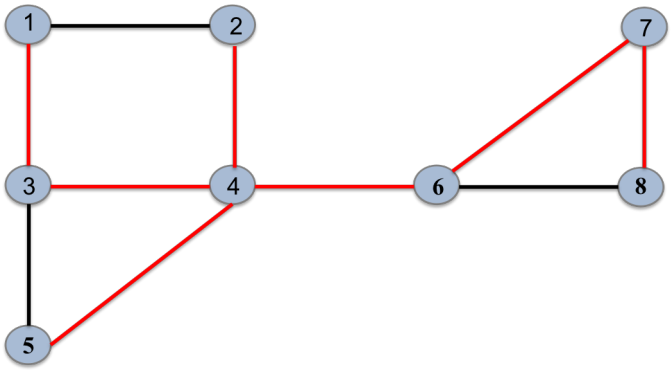


Figura A.7: Resultado final da Pesquisa em Profundidade

Anexo B

Algoritmo Húngaro

O algoritmo húngaro é usado para encontrar o emparelhamento ótimo num grafo bipartido completo; (Jungnickel, 2008) apresenta uma descrição detalhada deste algoritmo. Este algoritmo usa-se, por exemplo para resolver o problema de afetação descrito na secção 3.2.1, embora a descrição que será apresentada de seguida corresponde à resolução de um problema de maximização.
Um dos autores que propôs este algoritmo foi Kuhn [Kuh55,Kuh56]. Kuhn baseou-se nas ideias de König e Egorváry para criar este algoritmo daí ter-lhe dado o nome de "Algoritmo Húngaro".

Seja $G = (V, E)$ um grafo bipartido completo $K_{n,n}$, com $V = S \cup T$, onde $S = \{1, ..., n\}$ e $T = \{1^{'}, ..., n^{'}\}$, e uma matriz não negativa $W = (w_{ij})$: a entrada w_{ij} é o peso da aresta $(i, j^{'})$. Um par de vetores reais $u = \{u_1, ..., u_n\}$ e $v = \{v_1, ..., v_n\}$ é chamado um par nó-peso viável se a seguinte condição se verificar:

$$u_i + v_j \geq w_{ij} \qquad \forall \text{ i, j= 1,...,n} \tag{B.1}$$

O conjunto de todos os pares nó-peso viáveis (**u,v**) será denotado por **F** e o peso do emparelhamento ótimo por D.
O resultado que será apresentado de seguida é imediato através da soma B.1 sobre todas as arestas do emparelhamento M.

Lema B.1: *Para cada par nó-peso viável (**u,v**) e para cada emparelhamento perfeito M de G, tem-se que:*

$$W(M) \leqslant D \leqslant \sum_{i=1}^{n} (u_i + v_i) \tag{B.2}$$

Caso se consiga encontrar um par (**u,v**) viável e um emparelhamento M para os quais se verifica uma igualdade em B.2, então M tem de ser o emparelhamento ótimo. De facto, é sempre possível encontrar uma igualdade em B.2. O algoritmo Húngaro dará uma prova construtiva para esse facto. Agora é necessário analisar o caso da igualdade em B.2. Para um dado par (**u,v**) viável, seja $K_{u,v}$ o subgrafo de G com o conjunto de vértices V e as arestas $ij^{'}$ para as quais $u_i + v_j = w_{ij}$; H designa-se o subgrafo de igualdade para (**u,v**).

Lema B.2: *Seja $H = H_{u,v}$ o subgrafo de igualdade para o par (**u,v**) $\in$ **F**. Então tem-se*

que:

$$\sum_{i=1}^{n}(u_i + v_i) = D$$

se e só se H é um emparelhamento perfeito. Neste caso, todo o emparelhamento perfeito de H é um emparelhamento ótimo em (G, W).

Prova: *Primeiro considere-se que* $\sum_{i=1}^{n}(u_i + v_i) = D$ *e suponha-se que H não contém emparelhamento perfeito. Para* $J \subset S$, *denota-se por* $\Gamma(J)$ *o conjunto de todos os vértices* $j' \in T$ *que são adjacentes a algum vértice* $i \in J$. *Pelo Teorema 7.2.5 em (Jungnickel, 2008), existe um subconjunto J de S com* $|\Gamma(J)| < |J|$. *(Note-se que foram invertidos "os papeis"de S e T comparativamente com o Teorema 7.2.5). Coloca-se:*

$$\delta = min\{u_i + v_j - w_{ij} : i \in J, j' \notin \Gamma(J)\}$$

e define-se (u', v') *da seguinte forma:*

$$u_i' = \begin{cases} u_i - \delta \text{ para } i \in J \\ u_i \text{ para } i \notin J \end{cases} \qquad e \qquad v_j' = \begin{cases} v_j + \delta \text{ para } j' \in \Gamma(J) \\ v_j \text{ para } j' \notin \Gamma(J) \end{cases}$$

Então (u', v') *é novamente um par nó-peso viável: a condição* $u_i + v_j \geq w_{ij}$ *só poderia ser violada para* $i \in J$ *e* $j' \notin \Gamma(J)$. *Mas então* $\delta \leq u_i + v_j - w_{ij}$, *de modo que* $w_{ij} \leq (u_i - \delta)v_j = u_i' + v_j'$. *Obteve-se agora uma contradição:*

$$D \leq \sum(u_i' + v_j') = \sum(u_i + v_j) - \delta|J| + \delta|\Gamma(J)| = D - \delta|J| - |\Gamma(J)| < D$$

Reciprocamente, suponha-se que H contém um emparelhamento perfeito M. Então a igualdade B.1 é válida para cada aresta de M, e somando B.1 sobre todas as arestas de M produz-se a igualdade em B.2. Este argumento também mostra que todo o emparelhamento perfeito de H é um emparelhamento ótimo para (G, W).

O algoritmo Húngaro inicia-se com um par arbitrário $(\mathbf{u},\mathbf{v}) \in \mathbf{F}$; Geralmente considera-se:

$$v_1 = ... = v_n = 0 \quad \text{e} \quad u_i = max\{w_{ij} : j = 1, ..., n\} \quad \text{(para i=1,...,n)}$$

Se o subgrafo de igualdade correspondente contiver um emparelhamento perfeito, o problema está resolvido. Caso contrário, o algoritmo determina o subconjunto J de S com $|\Gamma(J)| < |J|$ e altera o par viável $(\mathbf{u},\mathbf{v})$ de acordo com a prova do **Lema B.2**. Isto diminui o valor do $\sum(u_i + v_j)$ e adiciona pelo menos uma aresta nova ij' com $i \in J$ e $j' \notin \Gamma(J)$ (relativamente ao subgrafo $H_{u,v}$) ao novo subgrafo de igualdade $H_{u',v'}$. Este procedimento é repetido até que o emparelhamento parcial deixe de ser máximo. Finalmente, obtém-se um grafo H que contém um emparelhamento perfeito M, que também é um emparelhamento ótimo de G. Para extender os emparelhamentos e alterar o par $(\mathbf{u},\mathbf{v})$, usa-se uma árvore alternada rotulada ("labelled alternating tree") apropriada em H. No algoritmo que se segue, guarda-se uma variável δ_j para cada $j' \in T$ que pode ser vista como um potencial: δ_j é o valor mínimo atual de $u_i + v_j - w_{ij}$. Além disso, $p(j)$ denota o primeiro vértice i para o qual foi obtido o valor mínimo.

Algoritmo B.1: Algoritmo Húngaro

Seja $G = (V, E)$ um grafo bipartido completo $K_{n,n}$ com $V = S \cup T$, onde $S = \{1, ..., n\}$ e $T = \{1^{'}, ..., n^{'}\}$, e com uma matriz "peso"não negativa $W = (w_{ij})$: a entrada w_{ij} é o peso da aresta $(i, j^{'})$. O algoritmo determina um emparelhamento ótimo em G descrito através de uma matriz "mate", onde para cada vértice V esta matriz contém a informação do vértice com o qual este está emparelhado. Note-se que Q no Algoritmo 1 representa o conjunto de vértices através do qual o algoritmo prossegue (não uma fila). Neste algoritmo também se usa um procedimento diferente designado "AUGMENT", que serve para criar um caminho aumentante (e alterar adequadamente o emparelhamento M) quando se encontra um vértice exposto. Um vértice exposto é um vértice que não pertence a nenhuma aresta emparelhada e um caminho aumentante é um caminho $p = [u_1, u_2, ..., u_k]$ no qual as arestas $[u_1, u_2], [u_3, u_4], ..., [u_{2j-1}, u_{2j}], ...$ são arestas livres e as arestas $[u_2, u_3], [u_4, u_5], ..., [u_{2j}, u_{2j+1}], ...$ são emparelhadas e os vértices u_1 e u_k são vértices expostos.

Algorithm 1 Algoritmo Húngaro

Input: n, w; $mate$

Output: Emparelhamento ótimo M

1: **Procedimento** HUNGARO(n, w; $mate$):
2: **for** $v \in$ V do $mate(v) \longleftarrow 0$
3: **for** $i = 1$ **to** n **do** $u_i \longleftarrow max\{w_{ij} : j = 1, ..., n\}$; $v_i \longleftarrow 0$
4: nrex$\longleftarrow n$
5: **while** nrex $\neq 0$ **do**
6: **for** $i = 1$ **to** n **do** $m(i) \longleftarrow false$; $p(i) \longleftarrow 0$; $\delta_i \longleftarrow \infty$
7: aug$\longleftarrow$ false; $Q \longleftarrow \{i \in S : mate(i) = 0\}$;
8: **repeat**
9: remover o menor vértice i de Q; $m(i) \longleftarrow$true; $j \longleftarrow 1$;
10: **while** aug=false **and** $j \leqslant n$ **do**
11: **if** $mate(i) \neq j^{'}$
12: **then if** $u_i + v_j - w_{ij} < \delta_j$
13: **then** $\delta_j \longleftarrow u_i + v_j - w_{ij}$; $p(j) \longleftarrow i$;
14: **if** $\delta_j = 0$
15: **then if** $mate(j^{'}) = 0$
16: **then** AUGMENT($mate, p, j^{'}; mate$);
17: aug$\longleftarrow$true; nrex$\longleftarrow n - 1$
18: **else** $Q \longleftarrow Q \cup \{mate(j^{'})\}$
19: **if** aug=false **and** $Q = \varnothing$
20: **then** $J \longleftarrow \{i \in S : m(i) = true\}$; $K \longleftarrow j^{'} \in T : \delta_j = 0\}$;
21: $\delta \longleftarrow min\{\delta_j : j^{'} \in T \setminus K\}$;
22: **for** $i \in J$ **do** $u_i \longleftarrow u_i - \delta$
23: **for** $j^{'} \in K$ **do** $v_j \longleftarrow v_j + \delta$
24: **for** $j^{'} \in T \setminus K$ **do** $\delta_j \longleftarrow \delta_j + \delta$
25: X $\longleftarrow \{j^{'} \in T \setminus K : \delta_j = 0\}$;
26: **if** $mate(j^{'}) \neq 0 \ \ \forall \ j^{'} \in$ X
27: **then for** $j^{'} \in$ X **do** $Q \longleftarrow Q \cup \{mate(j^{'})\}$
28: **else** escolhe-se $j^{'} \in$ X com $mate(j^{'}) = 0$;
29: AUGMENT($mate, p, j^{'}; mate$);
30: aug $\longleftarrow$ true; nrex $\longleftarrow$ nrex -1
31: **until** aug=true

32: **Procedimento** AUGMENT($mate$, p,$j^{'}$; $mate$):
33: **repeat**
34: $\longleftarrow p(j)$; $mate(j^{'}) \longleftarrow i$; seguinte $\longleftarrow mate(i)$; $mate(i) \longleftarrow j^{'}$;
35: **if** seguinte$\neq 0$ **then** $j^{'} \longleftarrow$ seguinte
36: **until** seguinte=0

Exemplo de Aplicação do Algoritmo Húngaro

Considere-se o grafo bipartido $K_{5,5}$ e a seguinte matriz de pesos W:

$$W = \begin{bmatrix} 3 & 8 & 9 & 1 & 6 \\ 1 & 4 & 1 & 5 & 5 \\ 7 & 2 & 7 & 9 & 2 \\ 3 & 1 & 6 & 8 & 8 \\ 2 & 6 & 3 & 6 & 2 \end{bmatrix}$$

Determine-se o emparelhamento ótimo deste grafo através do Algoritmo Húngaro.

Inicialmente, tem-se $n = 5$, $Q = \{1, 2, 3, 4, 5\}$ e a matriz *mate* possui todas as entradas nulas. Considera-se que o par viável (**u,v**) (de acordo com o passo (3) do algoritmo) é o seguinte:

$$\mathbf{v} = [0, 0, 0, 0, 0] \quad \text{e} \quad \mathbf{u} = [9, 5, 9, 8, 6].$$

Para executar o algoritmo, escolhe-se sempre o menor elemento de Q no passo (9). Sendo assim, na primeira fase considera-se $i = 1$ e obtém-se os seguintes valores de δ_j e $p(j)$:

$1'$	$2'$	$3'$	$4'$	$5'$	j'
6	1	0	∞	∞	δ_j
1	1	1	-	-	$p(j)$

Para que se torne clara a forma como ao longo do algoritmo se obtêm os valores de δ_j e $p(j)$, de seguida explica-se a forma como se obteve $\delta_1 = 6$ na primeira fase. Nesta fase do algoritmo tem-se $i = 1$ e $j = 1$ (passo (9)), como $1 \leq 5$ e $mate(1) \neq 1'$ (passos (10) e (11)) $\delta_1 = u_1 + v_1 - w_{11} = 9 + 0 - 3 = 6$ e $p(1) = 1$. Este processo é análogo para os outros vértices e para as outras fases do algoritmo.

Como $\delta_3 = 0$ e $mate(3') = 0$ o vértice $3'$ está exposto e portanto a aresta $\{1, 3'\}$ é a primeira aresta escolhida para o emparelhamento. Durante a segunda fase tem-se $i = 2$ e a aresta $\{2, 4'\}$ é adicionada ao emparelhamento. Na terceira fase, $Q = \{3, 4, 5\}$, por isso $i = 3$ e:

$1'$	$2'$	$3'$	$4'$	$5'$	j'
2	7	2	0	7	δ_j
3	3	3	3	3	$p(j)$

Como $4'$ já está saturado, o $mate(4') = 2$ é adicionado a Q. Em seguida, $i = 2$ é removido de Q, no passo (9), e obtém-se:

2	1	2	0	0	δ_j
3	2	3	3	2	$p(j)$

Neste momento, $5'$ é um vértice exposto e por isso executa-se o procedimento AUGMENT gerando um novo emparelhamento constituído pelas arestas $\{2, 5'\}$, $\{3, 4'\}$, e $\{1, 3'\}$, uma vez que anteriormente tinha-se $p(5) = 2$, $mate(2) = 4'$, $p(4) = 3$ e $mate(3) = 0$.
Durante a quarta fase, $Q = \{4, 5\}$; então $i = 4$ e:

$$\begin{array}{ccccccc} 5 & 7 & 2 & 0 & 0 & & \delta_j \\ 4 & 4 & 4 & 4 & 4 & & p(j) \end{array}$$

Dado que os vértices $4^{'}$ e $5^{'}$ estão ambos saturados, os "mates"correspondentes 3 e 2 são inseridos em Q. Com $i = 2$, $i = 3$ e $i = 5$ no passo (9), encontram-se os seguintes valores para δ_j e $p(j)$:

$$i = 2 \quad \begin{array}{ccccccc} 4 & 1 & 2 & 0 & 0 & & \delta_j \\ 2 & 2 & 4 & 4 & 4 & & p(j) \end{array}$$

$$i = 3 \quad \begin{array}{ccccccc} 2 & 1 & 2 & 0 & 0 & & \delta_j \\ 3 & 2 & 4 & 4 & 4 & & p(j) \end{array}$$

$$i = 4 \quad \begin{array}{ccccccc} 2 & 0 & 2 & 0 & 0 & & \delta_j \\ 3 & 5 & 4 & 4 & 4 & & p(j) \end{array}$$

Neste momento, tanto $2^{'}$ como $p(2) = 5$ estão expostos, por isso a aresta $\{5, 2^{'}\}$ é adicionada ao emparelhamento. A quarta fase termina; até agora encontrou-se o seguinte emparelhamento $M = \{\{1, 3^{'}\}, \{2, 5^{'}\}, \{3, 4^{'}\}, \{5, 2^{'}\}\}$ no subgrafo $H_{u,v}$, como se pode visualizar na Figura B.1.

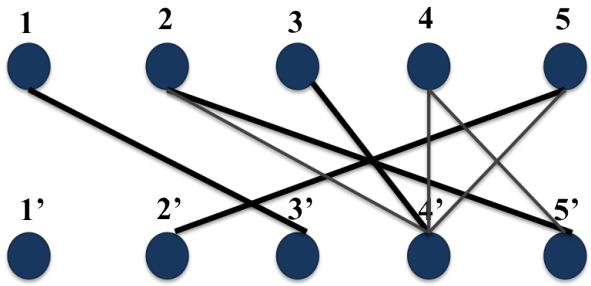


Figura B.1: Subgrafo de igualdade $H_{u,v}$ do exemplo do Algoritmo Húngaro

A quinta (e última) fase começa com $Q = \{4\}$, uma vez que os restantes vértices já estão emparelhados; então $i = 4$ e os valores de δ_j e $p(j)$ são os seguintes:

$$\begin{array}{ccccccc} 5 & 7 & 2 & 0 & 0 & & \delta_j \\ 4 & 4 & 4 & 4 & 4 & & p(j) \end{array}$$

Similar à fase anterior, 2 e 3 são inseridos em Q. Então os valores de δ_j e $p(j)$ são alterados para $i = 2$ e $i = 3$ da seguinte forma:

$$i = 2 \quad \begin{array}{ccccccc} 4 & 1 & 2 & 0 & 0 & & \delta_j \\ 2 & 2 & 4 & 4 & 4 & & p(j) \end{array}$$

$$i = 3 \quad \begin{matrix} 2 & 1 & 2 & 0 & 0 & \delta_j \\ 3 & 2 & 4 & 4 & 4 & p(j) \end{matrix}$$

Agora, pela primeira vez alcançou-se $Q = \varnothing$; sendo assim o par (**u,v**) viável é alterado de acordo com os passos (22) e (23). Com $J = \{2, 3, 4\}$, $K = \{4^{'}, 5^{'}\}$ e $\delta = 1$ obtém-se:

$$\begin{bmatrix} 3 & 8 & 9 & 1 & 6 \\ 1 & 4 & 1 & 5 & 5 \\ 7 & 2 & 7 & 9 & 2 \\ 3 & 1 & 6 & 8 & 8 \\ 2 & 6 & 3 & 6 & 2 \end{bmatrix}$$

$$\mathbf{v} = [0, 0, 0, 1, 1] \quad \text{e} \quad \mathbf{u} = [9, 4, 8, 7, 6]$$

Como exemplo de aplicação dos passos (22) e (23) considere-se o caso $i = 2$. Pelo passo (22) como $i = 2 \in J$ $u_2 = u_2 - \delta = 5 - 1 = 4$. No caso do passo (23) ao considerar-se $j^{'} = 4^{'}$ tem-se $v_4 = v_4 + \delta = 0 + 1 = 1$. Para os valores de i que não estejam contidos em J, u_i não se altera. O mesmo acontece com v_j para $j^{'} \notin K$. O processo é análogo para os restantes vértices e nas restantes fases do algoritmo.

O novo subgrafo de igualdade é representado na Figura B.2. Note-se que a aresta $\{5, 4^{'}\}$ foi removida do subgrafo de igualdade anterior, enquanto que a aresta $\{2, 2^{'}\}$ foi adicionada.

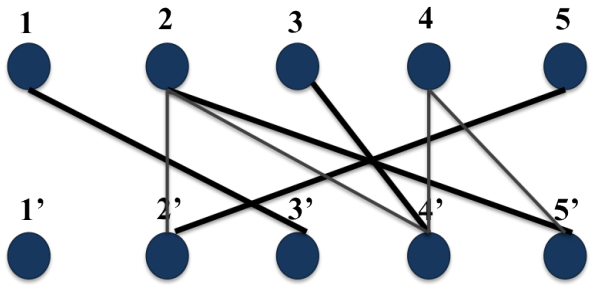


Figura B.2: Segundo Subgrafo de igualdade $H_{u,v}$ do exemplo do Algoritmo Húngaro

Seguidamente, o δ_j é atualizado no passo (24), sendo que os novos valores de δ_j são os seguintes:

$$\begin{matrix} 1 & 0 & 1 & 0 & 0 & \delta_j \\ 3 & 2 & 4 & 4 & 4 & p(j) \end{matrix}$$

Então pelo passo (25), $X = \{2^{'}\}$; como $2^{'}$ não está exposto e o $mate(2^{'}) = 5$ insere-se o vértice 5 em Q. Sendo assim tem-se $i = 5$ e os respetivos valores δ_j e $p(j)$ são:

$$\begin{matrix} 1 & 0 & 1 & 0 & 0 & \delta_j \\ 3 & 2 & 4 & 4 & 4 & p(j) \end{matrix}$$

Novamente, $Q = \varnothing$. Neste momento, o par viável (**u,v**) volta a ser alterado segundo os passos (22) e (23) com $J = \{2, 3, 4, 5\}$, $K = \{2^{'}, 4^{'}, 5^{'}\}$ e $\delta = 1$, obtendo o seguinte par:

$$\begin{bmatrix} 3 & 8 & \mathbf{9} & 1 & 6 \\ 1 & 4 & 1 & 5 & \mathbf{5} \\ \mathbf{7} & 2 & 7 & 9 & 2 \\ 3 & 1 & 6 & 8 & 8 \\ 2 & \mathbf{6} & 3 & 6 & 2 \end{bmatrix}$$

$$\mathbf{v} = [0, 1, 0, 2, 2] \quad \text{e} \quad \mathbf{u} = [9, 3, 7, 6, 5]$$

O subgrafo de igualdade correspondente a estas alterações está representado na Figura B.3: 3 arestas foram adicionadas e em contrapartida nenhuma foi removida.

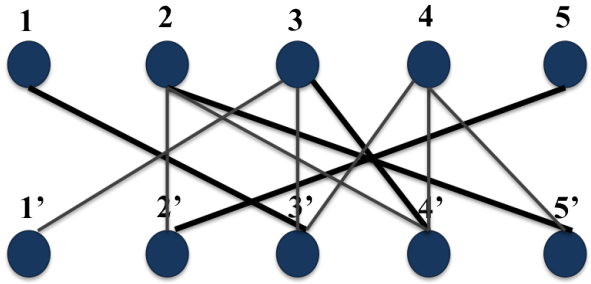


Figura B.3: Terceiro Subgrafo de Igualdade $H_{u,v}$ do exemplo do Algoritmo Húngaro

Assim δ_j foi alterado para:

0	0	0	0	0	δ_j
3	2	4	4	4	$p(j)$

Agora $X = \{1', 3'\}$. Como $1'$ está exposto, o emparelhamento pode ser aumentado. Com $p(1) = 3$, $mate(3) = 4'$, $p(4) = 4$ e $mate(4) = 0$ obtém-se o emparelhamento ótimo $M = \{\{1, 3'\}, \{2, 5'\}, \{3, 1'\}, \{4, 4'\}, \{5, 2'\}\}$ que está representado na Figura B.4 e que corresponde às entradas a negrito na matriz apresentada para o par viável $\mathbf{v} = [0, 1, 0, 2, 2]$ e $\mathbf{u} = [9, 3, 7, 6, 5]$; note-se que $W(M) = 35$ que na verdade é igual ao $\sum(u_i + v_i)$ para o par viável considerado naquele instante.

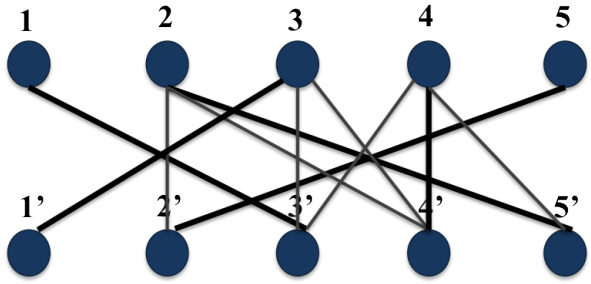


Figura B.4: Subgrafo de igualdade $H_{u,v}$ com emparelhamento perfeito do exemplo do Algoritmo Húngaro

Anexo C

Algoritmo de Edmonds

C.1 Algoritmo para o Emparelhamento num Grafo Bipartido

O problema de emparelhamento num grafo bipartido é um caso particular do problema de emparelhamento e consequentemente aplica-se o **Teorema 3.3.1.1**. Este problema será resolvido através de sucessivas descobertas de caminhos aumentantes p relativamente ao emparelhamento atual (em cada instante) através dos quais esse emparelhamento será aumentado através de $M \oplus P$.

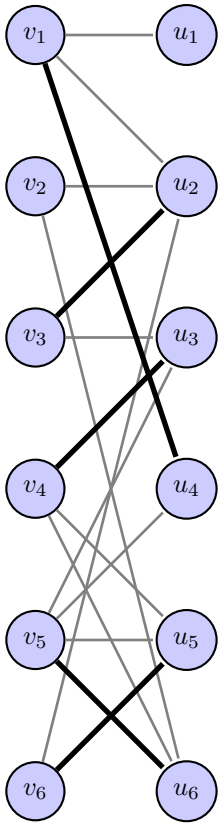


Figura C.1: Grafo Exemplo Algoritmo Emparelhamento Bipartido

Posto isto, o primeiro passo importante é descobrir como organizar a pesquisa do caminho aumentante p relativamente ao emparelhamento M num grafo bipartido $B = (V, U, E)$ de forma sistemática e eficiente. Para isso considere-se o exemplo representado na Figura C.1, onde está representado um grafo B e um emparelhamento M.

A pesquisa do caminho aumentante tem que começar pela construção de caminhos alternantes cujo vértice inicial seja um vértice exposto. Dado que um caminho aumentante tem de ter um vértice extremo em V e outro em U, não é perda de generalidade, começar a criar os caminhos alternantes apenas a partir dos vértices expostos de V (neste caso v_2).

Sendo assim considera-se o vértice exposto v_2 e pesquisam-se simultaneamente todos os caminhos alternantes possíveis, utilizando o método de Pesquisa em Largura. Começa-se no vértice v_2 e consideram-se todos os vértices adjacentes a v_2, nomeadamente u_2 e u_6 (Figura C.2 (a)). Visto que v_2 é um vértice exposto, todas as arestas adjacentes são livres. Pela definição de caminho alternante, agora é necessário olhar para as arestas emparelhadas provenientes de u_2 e u_6. Este passo é simples, uma vez que todos os vértices têm no máximo um "par". Naturalmente, se u_2 ou u_6 fossem expostos, a tarefa estaria concluída: ter-se-ia encontrado um caminho aumentante. Contudo, esse não é o caso. Então adicionam-se os vértices v_3 e v_5 ao conjunto dos caminhos alternantes, porque correspondem aos "pares"de u_2 ou u_6. Através desta construção v_3 e v_5 são vértices exteriores.

Continua-se a construção dos caminhos alternantes através de v_3 e v_5. Note-se que o vértice u_3 pode ser alcançado por v_3 ou v_5. Assume-se que u_3 é atingido primeiramente por v_3 e portanto a aresta $[v_5, u_3]$ é omitida. Obviamente que, ao fazer este raciocínio estão-se a perder caminhos aumentantes redundantes. Assim, descobrem-se novos vértices exteriores, nomeadamente v_4 através do vértice u_3 ,v_1 através do vértice u_4 (sendo u_4 alcançado através de v_5) e v_6 através de u_5 (sendo u_5 alcançado por v_5) (ver Figura C.2(a)). Finalmente nota-se que o vértice exterior v_1 é adjacente ao vértice exposto u_1. Sendo assim, descobriu-se um caminho aumentante, e através deste aumenta-se o emparelhamento M (ver Figura C.2(d)).

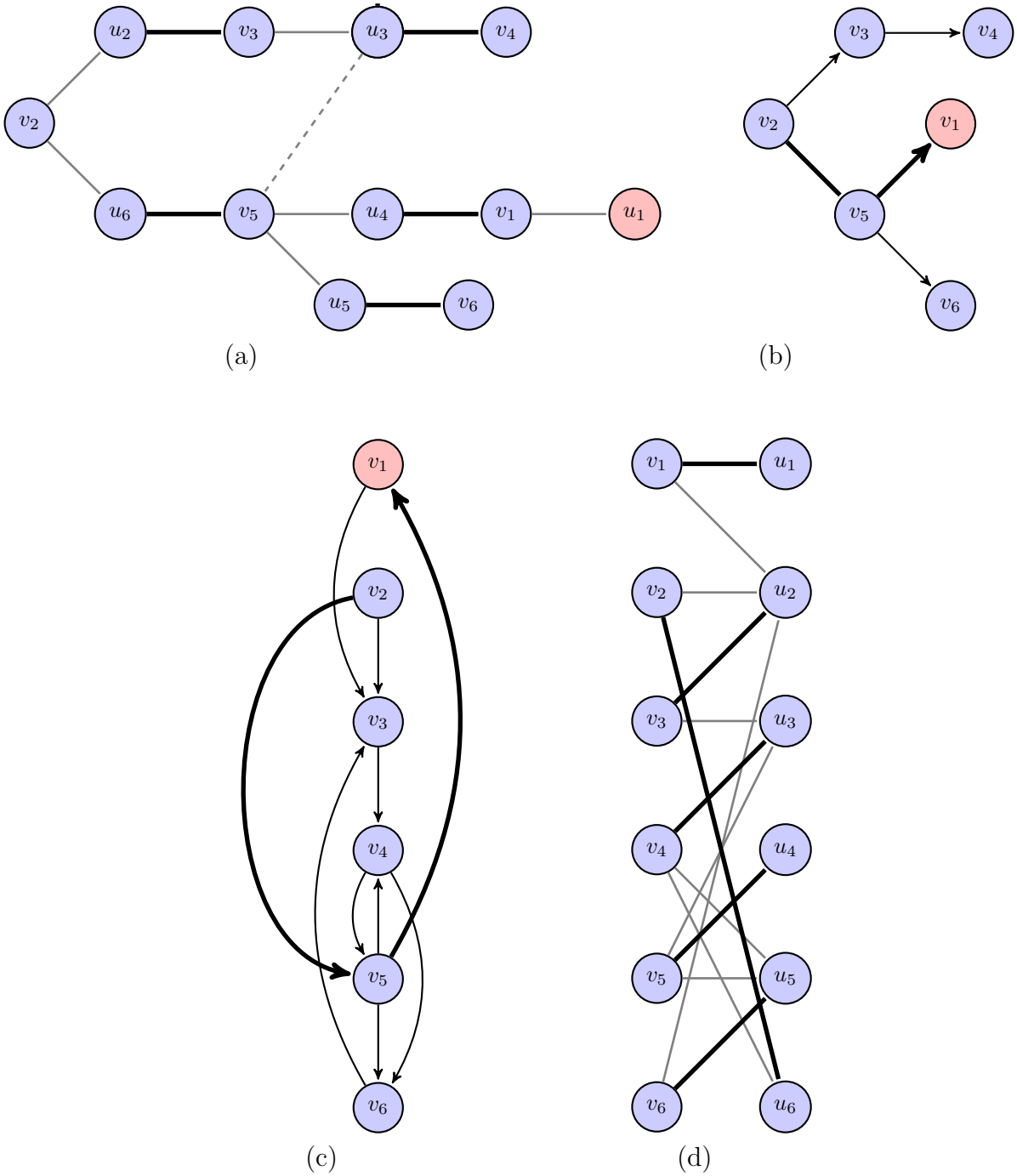


Figura C.2: Grafos ilustrativos das etapas do Algoritmo Bipartido para o Grafo da Figura C.1

O processo de pesquisa de caminhos aumentantes é bastante semelhante a uma pesquisa em largura. No entanto, ao recordar-se a construção dos caminhos alternantes nota-se que esta pesquisa em largura tem uma estrutura especial. As pesquisas de níveis ímpares (o vértice v_2 é o nível 0) são bastante triviais, porque o vértice seguinte é sempre o "par"do vértice

atual. Por exemplo no exemplo da Figura C.1 o vértice u_2 está numa posição ímpar da pesquisa e o vértice que o sucede é o seu "par", v_3.

Pode-se assim simplificar esta técnica de pesquisa ignorando os níveis ímpares e passando diretamente de um vértice exterior para o novo vértice exterior; no exemplo em questão a pesquisa desejada seria processada como está representada na Figura C.2(b). Obviamente que isto corresponde a procurar um digrafo (V, A), onde $(v_1, v_2) \in A$ se e só se v_2 pode ser o próximo vértice exterior após o vértice v_1, num caminho aumentante. Isto é, v_1 é adjacente ao "par"de v_2. O conjunto de vértices deste digrafo é V, porque claramente apenas os vértices de V podem tornar-se vértices exteriores num caminho alternante cujo vértice inicial pertence a V.

O digrafo auxiliar correspondente ao exemplo da Figura C.1 pode ser visto na figura C.2(c). A pesquisa dos caminhos aumentantes na Figura C.2(b) pode ser facilmente vista como uma pesquisa em largura no grafo auxiliar, iniciada no vértice v_2. Isto porque, para cada vértice são explorados todos os vértices adjacentes, antes de avançar para outro vértice. Mais concretamente, o processo inicia-se no vértice v_2. Para este vértice exploram-se os dois vértices adjacentes v_5 e v_3. Depois seleciona-se um desses vértices e volta-se a explorar todos os vértices adjacentes. Este processo repete-se até que seja descoberto um caminho aumentante ou até não existirem mais vértices para explorar.

O algoritmo para a resolução deste problema utiliza duas matrizes, *mate* e *exposto*, além da matriz *etiqueta* usada para a pesquisa. A matriz *mate* tem $|V| + |U|$ entradas e é usada para representar o emparelhamento atual; se um dado vértice v não estiver emparelhado $mate[v] = 0$. Para $v \in V$, $exposto[v]$ é um vértice $u \in U$ que está exposto e é adjacente a v; se não existir nenhum vértice nestas condições $exposto[v] = 0$. Obviamente que se um vértice $v \in V$ com $exposto[v] \neq 0$ é encontrado ao longo da pesquisa, significa que se encontrou um caminho aumentante.

O procedimento AUGMENT é recursivo. No exemplo em questão, primeiramente aplica-se o procedimento AUGMENT ao vértice v_1; e como $exposto[v_1] = u_1$, o procedimento AUGMENT altera o emparelhamento $[v_1, u_4]$ para $[v_1, u_1]$ e volta a efetuar o procedimento, mas desta vez aplicado ao vértice v_5, porque $etiqueta[v_1] = v_5$. Isto é, no digrafo auxiliar (Figura C.2(c)) o vértice v_1 é alcançado a partir de v_5; o emparelhamento $[v_5, u_6]$ é substituído por $[v_5, u_6]$ e volta-se aplicar o procedimento AUGMENT mas agora ao vértice v_2. Tem-se, $etiqueta[v_2] = 0$, porque v_2 é o vértice inicial da pesquisa. Então o procedimento AUGMENT emparelha o vértice v_2 com u_6 e para.

O algoritmo completo é descrito de seguida:

Algorithm 2 Algoritmo Edmonds Grafos Bipartidos

Input: Um grafo Bipartido $B = (V, U, E)$
Output: Emparelhamento máximo de B, representado na matriz *mate*

```
1: Procedimento EMPARELHAMENTO_BIPARTIDO (B = (V,U,E)):
2:   for v ∈ V ∪ U do mate[v] := 0
3: etapa:
4:     for all v ∈ V do exposto[v] := 0;
5:     A:= ⊘; ( iniciar a construção do digrafo auxiliar G = (V,A))
6:     for all [v,u] ∈ E do
7:     if mate[u] = 0 then exposto[v] := u else
8:       if mate[u] ≠ v then A := A ∪ (v, mate[u]);
9:       Q = ⊘;
10:      for all v ∈ V do if mate[v] = 0 Q = Q ∪ {v}, etiqueta[v] := 0;
11:    while Q ≠ ⊘ do
12:      seja v um vértice pertencente a Q;
13:      remover v de Q;
14:        if exposto[v] ≠ 0 AUGMENT(v), ir para  etapa;
15:        else
16:          for all v' não etiquetados tais que (v,v') ∈ A do
17:          etiqueta[v'] := v, Q = Q ∪ {v'};
18: Procedimento AUGMENT(v):
19:    if etiqueta[v] = 0 then mate[v] := exposto[v],
20:                mate[exposto[v]] := v;
21:    else
22:      exposto[etiqueta[v]] := mate[v];
23:      mate[v] := exposto[v];
24:      mate[exposto[v]] := v;
25:      AUGMENT(etiqueta[v])
```

Pelo **Teorema 10.3** descrito em (Papadimitriou and Kenneth, 1998), este algoritmo resolve corretamente o problema de emparelhamento para um grafo bipartido em tempo $O(\min(|V|^{\frac{1}{2}}.|E|)$.

C.2 Emparelhamento em Grafos não Bipartidos: Flores

O **Teorema C.1.1** é na mesma válido para os problemas de emparelhamento em grafos gerais, e portanto o método adotado na **Secção C.1** (começar com um emparelhamento vazio e repetidamente descobrir caminhos aumentantes através dos quais o emparelhamento será sucessivamente aumentado) pode ser também aplicado ao problema de emparelhamento

em grafos não bipartidos. Infelizmente, o facto de neste problema não existir uma estrutura bipartida torna a tarefa de encontrar caminhos aumentantes muito mais difícil.

Examine-se primeiro como é que a técnica do digrafo auxiliar da **Secção C.1** pode ser modificada de forma a funcionar para grafos gerais. Primeiro, no caso bipartido, sabia-se que os vértices pertencentes a U nunca podiam ser vértices exteriores num caminho alternante cujo vértice inicial pertencia a V. Consequentemente, o digrafo auxiliar apenas possuía vértices do conjunto V. Neste caso (grafos gerais), o digrafo auxiliar tem que conter todos os vértices.

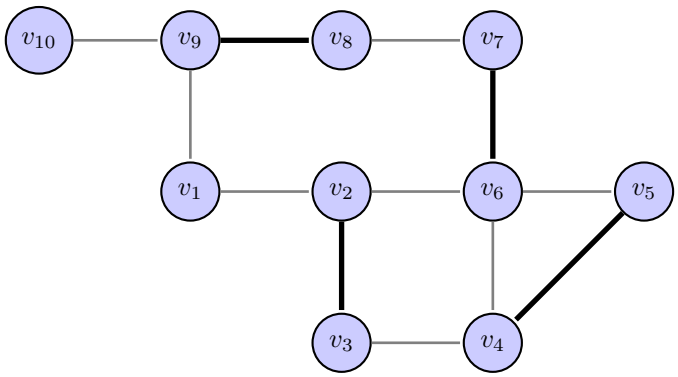
Os arcos do digrafo auxiliar representam informação pertinente: um arco (u, v) significa que u pode ser o próximo vértice exterior num caminho alternante onde v é um vértice exterior. Consequentemente, caminhos aumentantes correspondem a caminhos no digrafo auxiliar que vão de um vértice exposto até um vértice alvo (que é um vértice v com $exposto[v] \neq 0$), exatamente como se fez no caso bipartido.

O que torna o problema de emparelhamento em grafos gerais mais complicado, é que podem existir caminhos no digrafo auxiliar com início num vértice exposto e fim num vértice alvo que não correspondam a caminhos aumentantes relativamente ao emparelhamento atual. Veja-se um exemplo onde esta situação acontece. Considere-se o grafo G representado na Figura C.3(a) e o emparelhamento M também ilustrado na mesma figura. Neste momento M não é máximo, porque existe um caminho aumentante relativamente a M em G, nomeadamente, $p = [v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}]$. Naturalmente, p corresponde ao caminho $p' = [v_1, v_3, v_5, v_7, v_9]$ no digrafo auxiliar relativo a M, representado na Figura C.3(b).

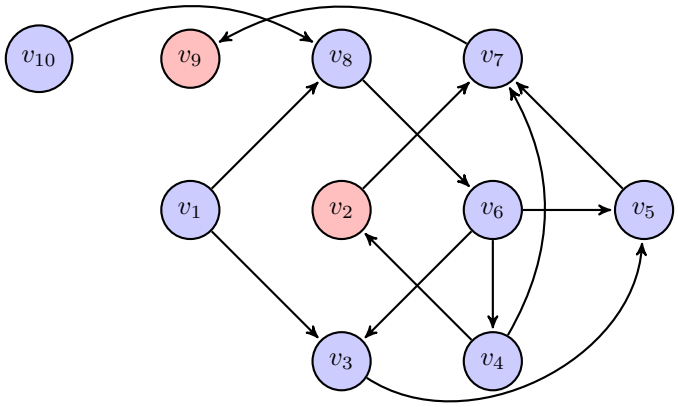
Porém, existem caminhos no digrafo auxiliar que não correspondem a um caminho aumentante legítimo. Por exemplo, considere-se o caminho $q' = [v_1, v_8, v_6, v_5, v_7, v_9]$. Não existe um caminho aumentante em G cuja sequência de vértices tem como vértices exteriores os vértices pertencentes a q'. Caso se tente a correspondência $(u_1, u_2, ..., u_k) \leftrightarrow [u_1, mate[u_2], u_2, ..., mate[u_k], u_k, exposto[u_k]]$ - através da qual se consegue restabelecer um caminho aumentante no caso bipartido - obtém-se o *"passeio"*
$q = [v_1, v_9, v_8, v_7, v_6, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}]$. Um **passeio** num grafo é uma sequência de vértices onde cada dois vértices consecutivos são ligados por um arco. No caso do passeio q sabe-se que existem os arcos (v_1, v_9), (v_8, v_7), (v_6, v_4), (v_5, v_6), (v_7, v_8) e (v_9, v_{10}). Naturalmente, q não é um caminho aumentante; de facto nem é um caminho, porque existem vértices e arestas repetidos em q, tais como as arestas emparelhadas $[v_8, v_9]$ e $[v_6, v_7]$. Estas repetições não estão refletidas em q', porque em cada instante diferentes pontos extremos das arestas emparelhadas são utilizados como vértices exteriores. Igualmente, $M \oplus Q$ (onde Q é o conjunto das arestas do passeio q) também não é um emparelhamento.

Uma vez que a técnica do digrafo auxiliar funcionou com sucesso para grafos bipartidos e os circuitos ímpares são a única característica que não está presente nos grafos bipartidos, existe a suspeita de que são estes circuitos que fazem com que a técnica não funcione eficientemente nestes casos (grafos gerais). E é exatamente isso que acontece. Até se encontrar um circuito ímpar $c = [v_6, v_4, v_5, v_6]$ (Figura C.3), q é um caminho alternante perfeitamente legítimo. Após a descoberta do circuito, q atravessa o circuito e começa a "entrelaçar-se"sobre si mesmo, causando as consequências referidas em cima. O conceito "entrelaçar-se"é utilizado no sentido de representar a situação em que o caminho q depois de atravessar o circuito c

volta a visitar os vértices anteriormente visitados nomeadamente, os vértices v_7, v_8, v_9.



(a)



(b)

Figura C.3: Grafo G e o seu respetivo Digrafo Auxiliar

Nem todos os circuitos ímpares causam um comportamento indesejável. Por exemplo, o circuito com 9 vértices no grafo da Figura C.4 não conduz o algoritmo do digrafo auxiliar a falsos caminhos aumentantes. Isto acontece porque, intuitivamente, este circuito é muito "esparso"em arestas emparelhadas, e portanto o caminho aumentante não consegue atravessá-lo e "entrelaçar-se". Os circuitos ímpares suspeitos (aqueles que podem levar a falsos caminhos aumentantes) são aqueles que possuem $2k + 1$ vértices e k arestas emparelhadas. Estes circuitos são designados por *flores*. Note-se que, como na maioria da terminologia do emparelhamento, uma *flor* é definida só no contexto de algum emparelhamento fixado.

Deteção de Flores

A causa da falha desta técnica do digrafo auxiliar foi detetada: é a presença de flores. Sendo assim, se um grafo não possuir flores relativamente ao emparelhamento atual, a pesquisa de caminhos aumentantes pode ser feita como no caso bipartido. Consequentemente, é necessário modificar o algoritmo para que este funcione eficientemente para grafos gerais

(grafos que podem ter flores). Isto é, é necessário que este em primeiro lugar consiga detetar a existência de flores, e em segundo consiga também encontrar caminhos aumentantes mesmo nos casos em que estas existem. Para manter o algoritmo simples, vai-se procurar um caminho aumentante com origem num vértice exposto de cada vez, em vez de se procurar para todos os vértices expostos em simultâneo, como aconteceu no caso bipartido.

Uma flor torna-se relevante para o algoritmo quando é descoberta pela primeira vez; pois é quando todos os seus vértices são vértices exteriores ou "pares"de vértices exteriores. Esta situação pode ser analisada na Figura C.5(b) para a flor ilustrada na Figura C.5(a). O que acontece neste momento na pesquisa é que o próximo vértice a ser explorado é o u_{2j} (figura C.5(b)) que é o "par"do vértice u_{2j-1}. Mas, u_{2j-1} já é um vértice exterior. Isto significa que a pesquisa tenta usar a aresta emparelhada $[u_{2j}, u_{2j-1}]$ duas vezes e, sendo assim, descobriu-se uma flor.

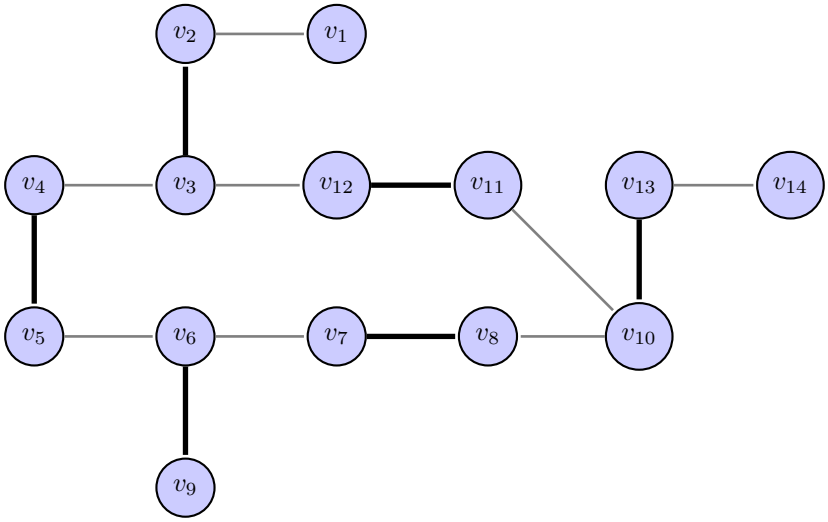
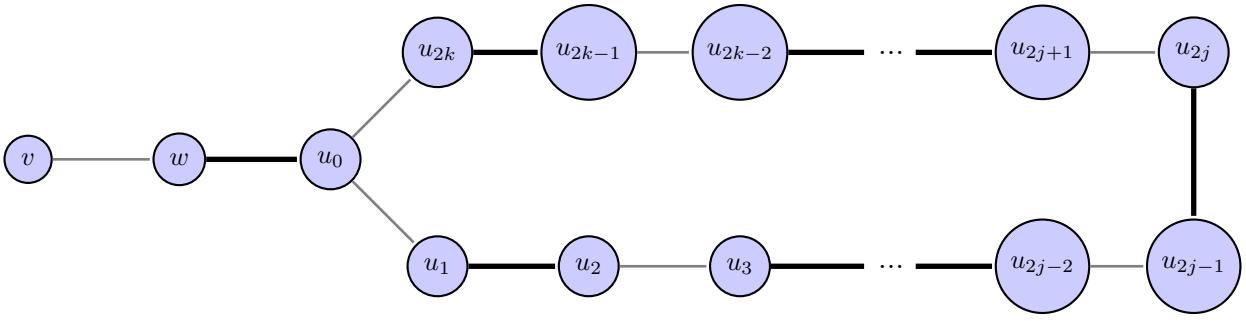
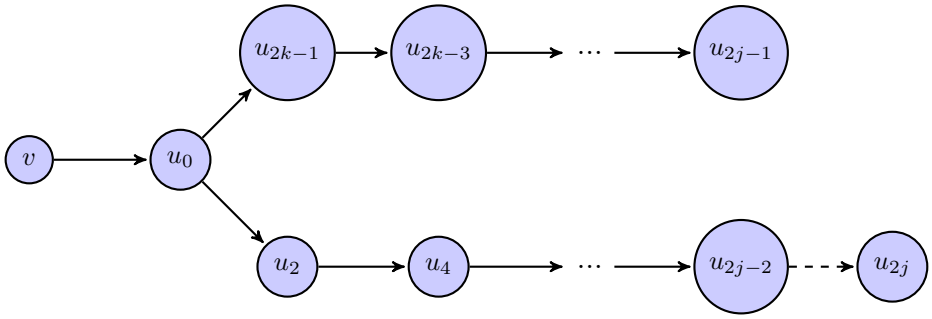


Figura C.4: Grafo com o circuito de 9 vértices



(a)



(b)

Figura C.5: Grafos Auxiliares na explicação da noção de Flor

Para encontrar todos os vértices que pertencem à flor não é muito difícil: "Anda-se para Trás"(backtrack) tanto no vértice u_{2j} como no vértice u_{2j-1} para encontrar os dois caminhos existentes do vértice exposto v até cada um dos vértices em questão. Encontra-se o último vértice que estes caminhos têm em comum. Esse vértice, u_0 no caso do exemplo das Figuras C.5(a) e C.5(b), é a base da flor; isto é u_0 é o único vértice da flor que não está emparelhado com outro vértice da flor. A flor é então constituída por todos os vértices exteriores dos caminhos de u_0 para u_{2j} e u_{2j-1} juntamente com os respetivos "pares"(exceto o de u_0). Ou seja, u_0, u_1, u_2, u_3,..., u_{2j-2}, u_{2j-1}, u_{2k}, u_{2k-1}, u_{2k-2}, ..., u_{2j+1} e u_{2j}.

Pesquisa de caminhos aumentantes

O processo anteriormente descrito abrange uma das duas extensões que o digrafo auxiliar tem que possuir para que este funcione eficientemente em grafos gerais: nomeadamente a deteção de flores. Após a deteção de uma flor é necessário definir como se prossegue a pesquisa de um caminho aumentante. Uma ideia simples é fazer desaparecer a flor do grafo através do encolhimento desta, isto é, substituir a flor por um único vértice.

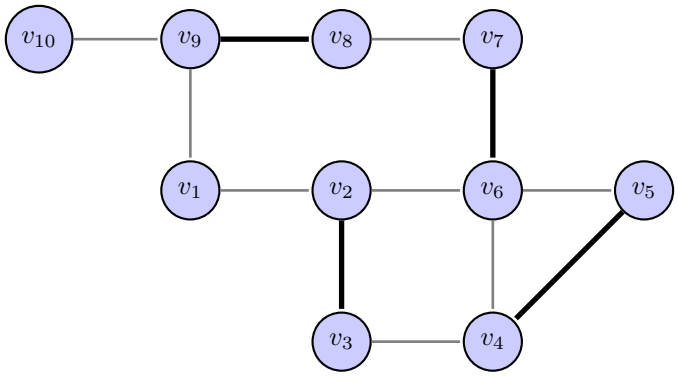


Figura C.6: Grafo Exemplo onde existe uma flor b

Formalmente, a noção de encolhimento é: se b é uma flor num grafo $G(V, E)$ associada a um emparelhamento M, então o grafo resultante a partir de G pelo encolhimento de b é $G/B = (V/b, E/B)$, onde V/b é V com todos os vértices de b omitidos e um novo vértice v_b; e E/b é E com todas as arestas com ambas as extremidades em b omitidas e as arestas $[v, u]$ tal que $u \in b$ e $v \notin b$ substituídas por $[v, v_b]$. Por exemplo, para o grafo G e a flor

$b = \{v_4, v_5, v_6\}$ representados na Figura C.6, o grafo G/b resultante do encolhimento da flor b está representado na Figura C.7(a), juntamentamente com o encolhimento da flor $v_{b'} = \{v_2, v_3, v_b\}$ no grafo G/b C.7(b).

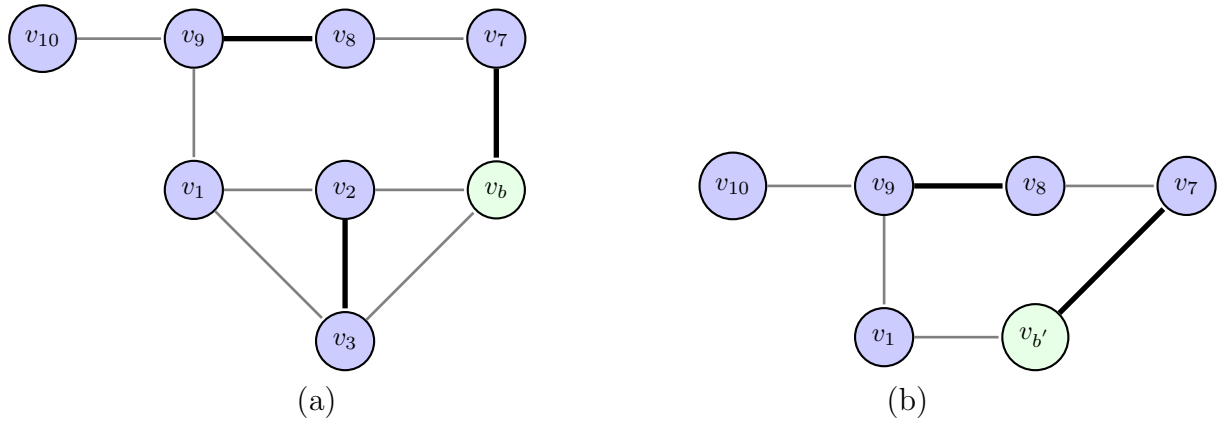


Figura C.7: Encolhimento da flor v_b e da flor $v_{b'}$

Naturalmente, para o encolhimento representar uma operação válida, deve-se mostrar que não adiciona nem remove caminhos aumentantes.

Lema C.2.1: *Suponha-se que enquanto se procura um caminho aumentante através de um vértice exposto u num grafo G relativamente a M, descobre-se uma flor b. Então existe um caminho alternante de u para qualquer vértice de b cuja aresta final é uma aresta emparelhada.*

Prova: *Uma vez que b foi descoberta a partir de u existe um caminho alternante de u até ao vértice u_0 (base de b), como se pode ver na Figura C.8 . Então um vértice v de b é alcançado a partir de u através de um dos caminhos alternantes (ver Figura C.8). Um destes caminhos termina com aresta emparelhada.*

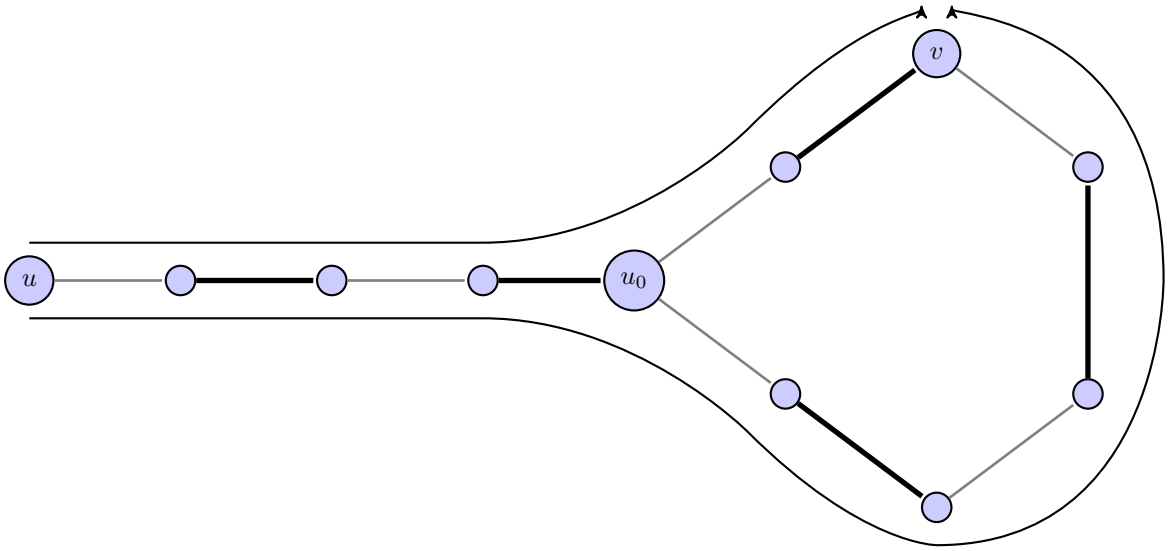


Figura C.8: Figura Auxiliar na prova do Lema C.2.1

Considere-se que G é um grafo, M um emparelhamento e b a flor em G associada a M. Então M/b é o emparelhamento que resulta de M omitindo todas as arestas emparelhadas de b e alterando qualquer aresta emparelhada da forma $[v, u]$ com $u \in b$ por uma aresta da forma $[v, v_b]$.

Teorema C.2.1: *Suponha-se que enquanto se pesquisa um caminho aumentante de um vértice u no grafo relativamente a um emparelhamento M, descobre-se uma flor b. Então existe um caminho aumentante a partir de u em G se e só se existir um caminho aumentante a partir de u (v_b se u for a base de b) em G/b relativamente a M/b.*

Prova: *Se se supuser que existe um caminho aumentante p a partir de u em G/b relativamente a M/b, se p não passar por v_b, então p é um caminho aumentante em G. Se p passar por v_b, pode-se escrever $p = [u, p', w, v_b, w', p'']$ (u, w são os vértices extremos do caminho p' por isso é que aparece $[u, p', w, ..])$, onde p' e p'' são caminhos. Uma das arestas $[w, v_b]$, $[w', v_b]$ é uma aresta emparelhada. Suponha-se que $[w, v_b]$ é emparelhada (o outro caso é análogo). Então $[w, u_0] \in M$ e $[w', u_j] \in E - M$ para a base u_o e o outro vértice u_j de b (Figura C.9). O caminho aumentante com vértice inicial u em G é então $p = [u, p', w, u_0, p''', u_j, w', p'']$ onde p''' é o caminho de u_0 até u_j e termina com uma aresta emparelhada (no caso de u_j=u_0 tem-se o caminho vazio). Se $u = v_b$ o argumento é bastante similar.*

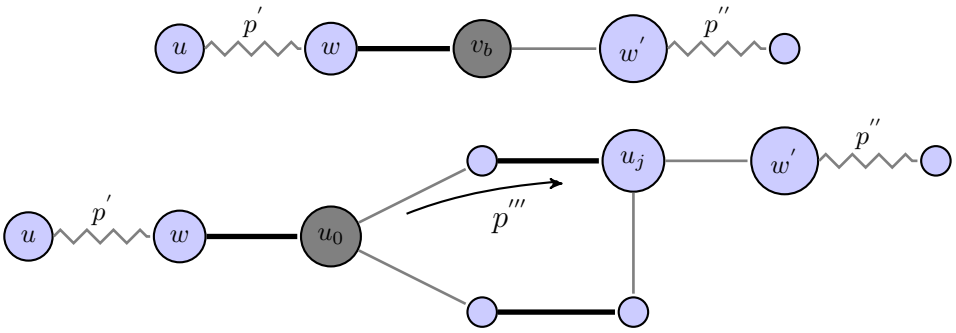


Figura C.9: Figura Auxiliar na prova do Teorema C.2.1

Suponha-se agora que existe um caminho aumentante p a partir de u em G relativamente a M. Novamente, se p não passa em qualquer vértice da flor b, está provado. Caso contrário, existem dois casos:

Caso 1: *Considere-se que p entra em b no vértice base u_0 através de uma aresta emparelhada (Figura C.10(a)). Seja u_j o último vértice de b no caminho, se u_j=u_0 está provado porque, p é também um caminho aumentante em G/b com u_0 substituído por v_b.Caso contrário, $p = [u, p', u_0, p'', u_j, w, p''']$ e $[u_j, w] \notin M$. Então $p = [u, p', v_b, w, p''']$ é um caminho aumentante em G/b.*

Caso 2: *Suponha-se que p entra em b através de uma aresta livre. Distinguem-se dois subcasos:*

SubCaso 2(a): *Considere-se que p sai de b pela base através de uma aresta emparelhada.*

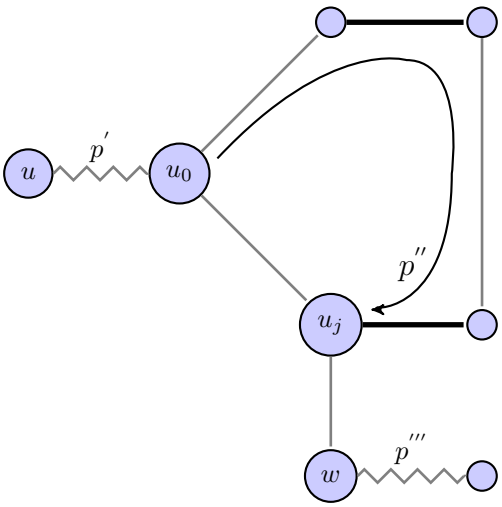
Este caso é similar ao ***Caso 1****.*

SubCaso 2(b): *Na situação em que p sai de b através de uma aresta livre. Então* $p = [u, p', u_i, p'', u_j, w, p''']$ *(ver Figura C.10(b)). Pelo* ***Lema C.2.1****, existe um caminho alternante q, a partir de u até à base* u_0 *de b, que termina com uma aresta emparelhada. É preciso considerar-se 3 possibilidades.*

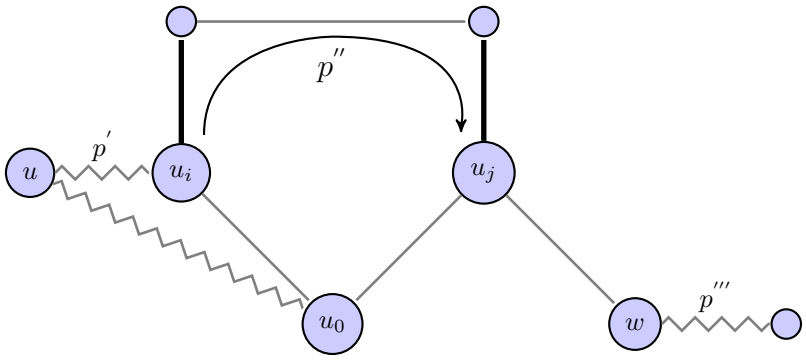
1) *Primeiro supõe-se que* p''' *e q não se intersetam. Então* $[u, q, v_b, w, p''']$ *é um caminho aumentante.*

2) *Suponha-se agora que* p''' *e q se intersetam (Figura C.10(c)). Então* $q = [u, q', x, q'', u_0]$ *e* $p''' = [p^{(4)}, x, p^{(5)}]$*, onde* $p^{(5)}$ *não tem nenhum vértice em comum com q. Então* $[u, p', v_b, q'', x, p^{(5)}]$ *é um caminho aumentante, onde se assume que* q'' *e* p' *não têm vértices em comum.*

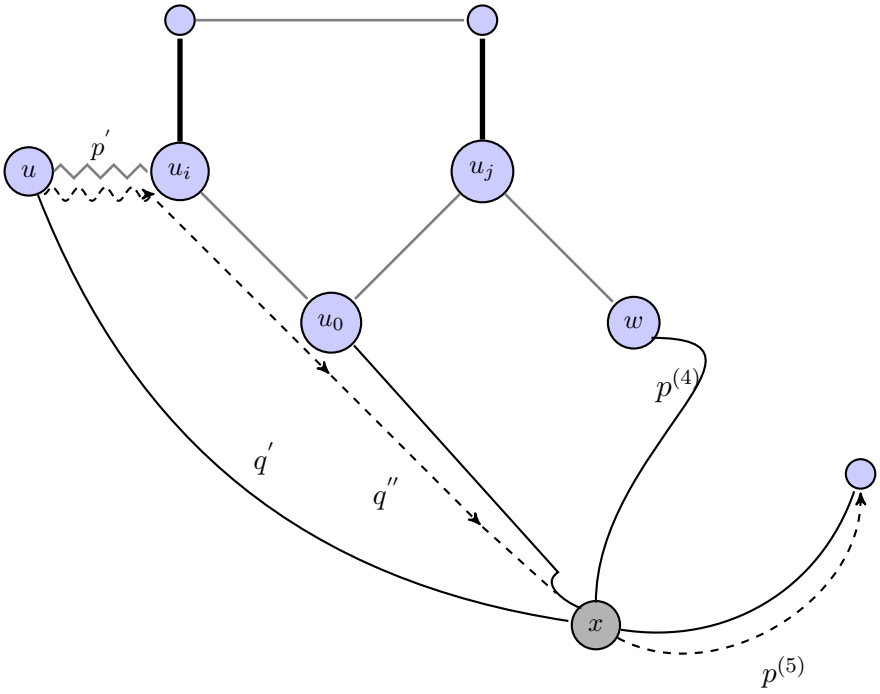
3) *Este é o mesmo que em* ***(2)****, exceto que* q'' *e* p' *intersetam-se (Figuras C.10(d) e (e)). Seja Y o último vértice de* q'' *que está em* p' *ou em* p''*. Se Y está em* q'''*, então constrói-se um caminho aumentante em G/b usando p até* v_b *q volta até Y e* p'''*, como pode ser visto na Figura C.10(d). Se Y está em* p'*, então usa-se* p' *até Y, que até* v_b *e* p'''*, como está representado na Figura C.10(e). O teorema está provado.*



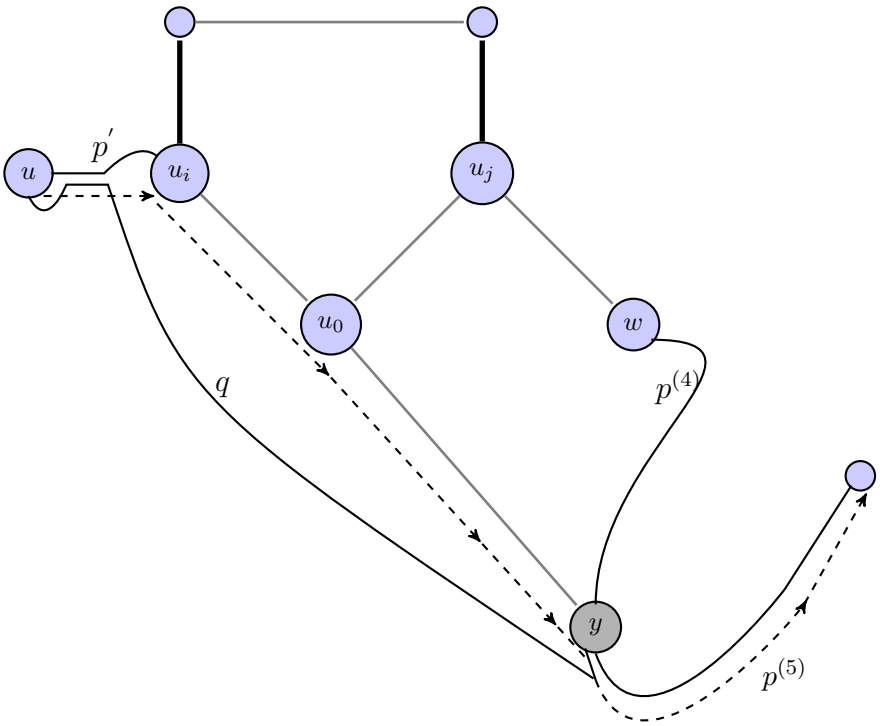
(a)



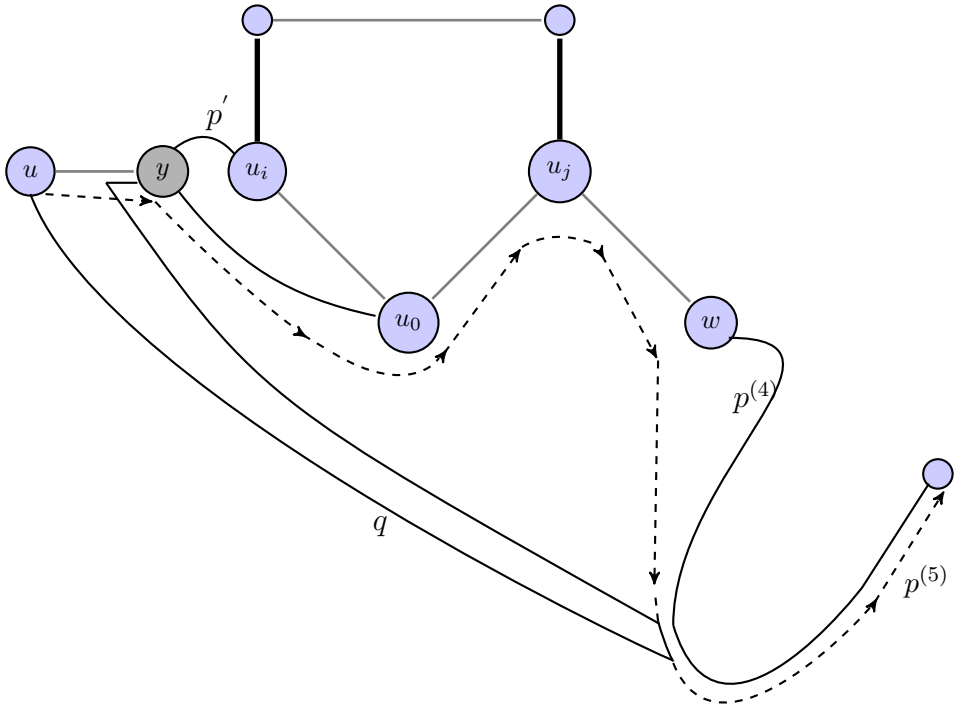
(b)



(c)



(d)



(e)

Figura C.10: Figura Auxiliar na prova do Teorema C.2.1
casos 1 e 2

C.3 Emparelhamento em Grafos não Bipartidos: Algoritmo

Nesta secção, será descrito um algoritmo para o problema de emparelhamento considerando grafos gerais. Este algoritmo é o algoritmo para o caso do emparelhamento em grafos bipartidos, juntamente com as características e os pormenores necessários para que este seja eficiente quando se depara com uma flor.

Começa-se com o emparelhamento vazio e repetidamente escolhe-se um vértice exposto, a fim de através deste procurar um caminho aumentante. Para simplificar, procura-se um caminho aumentante para um vértice exposto de cada vez, e não para todos os vértices expostos, como foi feito no caso bipartido. O método de pesquisa irá novamente utilizar um digrafo auxiliar.

Suponha-se que a um dado momento não se consegue encontrar um caminho aumentante a partir de um vértice exposto u. Então u não tem qualquer utilidade nesta fase. Pelo Teorema apresentado de seguida, u será uma má escolha como ponto de partida em todas as fases seguintes.

Teorema C.3.1: *Suponha-se que num grafo G não existe nenhum caminho aumentante*

cujo vértice inicial é um vértice exposto u, relativamente ao emparelhamento M. Seja p um caminho aumentante tendo como vértices extremos outros dois vértices expostos v e w. Então também não existe nenhum caminho aumentante a partir de u relativamente ao emparelhamento $M \oplus P$.

Prova: *Considere-se que existe um caminho aumentante q a partir de u relativamente a $M \oplus P$. Se q não tem nenhum vértice em comum com p, então o aumento obtido através de p não muda nada que afete q e portanto q era um caminho aumentante a partir de u antes do aumento, contrariamente à hipótese inicial.*
Então seja $q = [u_1 \equiv u, u_2, ..., u_k \equiv u']$ e seja u_j o primeiro vértice de q comum a p (ver Figura C.11). Uma das partições em que u_j divide p tem que terminar em u_j com uma aresta de M; esta partição, juntamente com a partição de q até u_j constituem um caminho aumentante a partir de u relativamente a M, contrariamente à hipótese inicial.

Corolário *Se em alguma fase não existir um caminho aumentante a partir de u, então nunca irá existir um caminho aumentante a partir de u.*

Prova: *É fácil provar por indução matemática em k e usando o* ***Teorema C.3.1****, que para todo o k efetuando k aumentos após ter-se falhado a descoberta de um caminho aumentante a partir de u, continuará a não existir nenhum caminho aumentante a partir de u.*

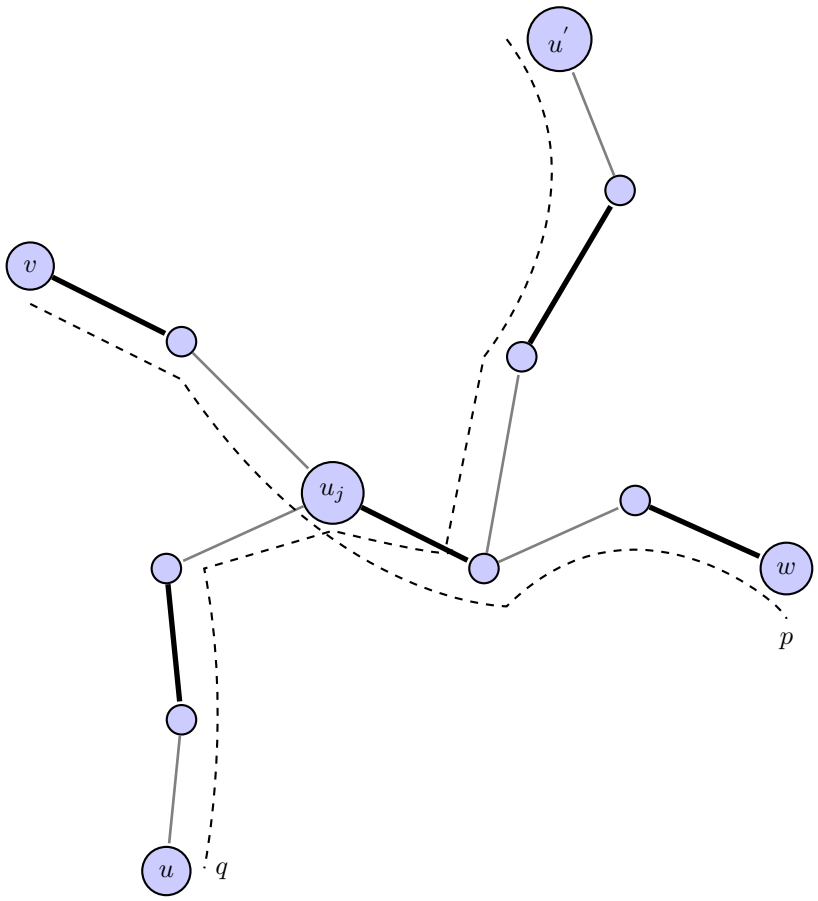


Figura C.11: Prova do Teorema C.3. Arestas a negrito pertencem a $M \oplus P$

Portanto, quando a pesquisa para encontrar um caminho aumentante com vértice inicial u falhar, o vértice u nunca mais volta a ser considerado como potencial vértice de partida para um caminho aumentante. Para que tal aconteça, no algoritmo em questão existe uma matriz "considerado", inicialmente definida como sendo a matriz nula. Se um vértice u é usado como vértice inicial para a pesquisa do caminho aumentante, $considerado[u] = 1$, o que implica qu este vértice não será mais considerado.

Pesquisa de flores

Como no caso bipartido, o digrafo auxiliar (V, A) é a ajuda básica para a realização da pesquisa. Além disso, a matriz *exposto* será novamente utilizada para identifcar vértices alvo - isto é vértices que sejam adjacentes a um vértice exposto diferente do que está a ser considerado na pesquisa. A fim de ser capaz de detetar flores, este algoritmo possui uma matriz designada *visto* com $|V|$ entradas, inicialmente todas com o valor nulo; quando $visto[v] = 1$ significa que v é o "par"de um vértice exterior; se v é considerado como vértice exterior em algum momento na pesquisa, ambas as extremidades da aresta $[v, mate[v]]$ são vértices exteriores, e portanto foi descoberta uma flor.

Assim que a primeira flor é encontrada, esta é encolhida. De acordo com o **Teorema C.2.1**, isto é uma operação racional, porque preserva a existência de caminhos aumentantes a partir do vértice que está a ser considerado. Assim, dá-se à flor atual o nome b e o grafo atual G transforma-se em G/b. Considere-se o exemplo da Figura C.6, pode-se visualizar G/b na Figura C.7(a). Subsequente à descoberta de b, descobriu-se uma outra flor $v_{b'} = \{v_2, v_3, v_b\}$ (Figura C.7)(a). Esta segunda flor foi também encolhida e portanto obtém-se um outro grafo $G/b/b^{'}$ (Figura C.7)(b) que passa a ser o grafo atual, e assim sucessivamente.

Procedimento "blossom" e encolhimento

Depois de cada descoberta de uma flor b, é necessário guardar alguma informação útil acerca desta relacionada com o seu encolhimento. Todo esse processo é feito através de um procedimento designado *blossom* inserido no Algoritmo 3. O que o procedimento faz é o seguinte:

1) Encontra todos os nós de b através de "backtracking"na matriz *etiqueta* até ao primeiro vértice em comum nos dois caminhos, a base da flor (Figura C.12). Os vértices de b são exatamente os vértices dos dois caminhos e os seus respetivos "pares". Guarda-se a informação que v pertence a b definindo $blossom[v] = b$ - note-se que v pode ser ele mesmo um vértice flor. É também guardada a base de b e a ordem cíclica exata na qual os vértices de b ocorreram.

No exemplo representado na Figura C.12 o "backtracking"é efetuado a partir dos vértices 4 e 5. O caminho obtido através do vértice 4 é $4 - 8 - 6-$base de b, enquanto que o obtido através de 5 é $5 - 3-$base de b. Encontra-se assim a base de b, que neste caso está denotada

por " base de b"e os vértices pertencentes à flor b [base de $b, 1, 2, 3, 4, 5, 6, 7, 8$].

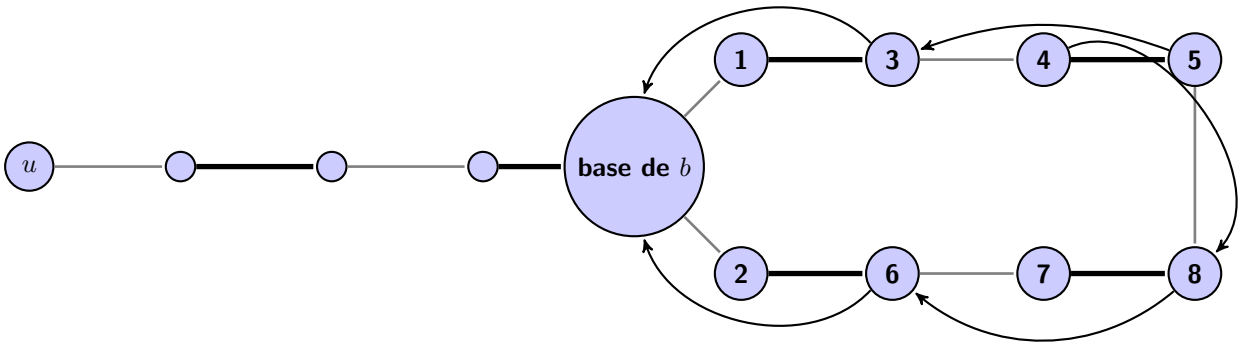


Figura C.12: Determinação dos vértices pertencentes a uma flor

Depois de identificados os vértices de b, é preciso verificar se existe algum vértice u_j entre estes, tal que $exposto[u_j] \neq 0$. Se assim for, deve-se aumentar o emparelhamento a partir de v_b.

2) Para todo o vértice x pertencente a b, no digrafo auxiliar A, na fila de espera Q e na matriz *etiqueta* substitui-se x pelo novo vértice v_b. Este é o efeito causado pelo encolhimento dos vértices de b num único vértice.
No caso concreto do exemplo da Figura C.12 os vértices [base de $b, 1, 2, 3, 4, 5, 6, 7, 8$] eram substituídos por v_b.

O procedimento de pesquisa, descoberta e encolhimento de flores continua até que no grafo atual seja descoberto um caminho aumentante a partir de u - ou a partir de v_b, onde v_b é a mais recente flor encolhida que contém u - ou até não se conseguir encontrar nenhum caminho ou flor na pesquisa no digrafo auxiliar.

Criação de caminhos aumentantes no grafo inicial

Se é encontrado um caminho aumentante p no grafo atual, é necessário através deste criar um caminho aumentante no grafo original. Este passo pode não ser trivial uma vez que pode conter muitos vértices flor e portanto não pode ser usado diretamente para aumentar o emparelhamento (é necessário realizar algumas transformações). Considere-se, por exemplo, o caminho aumentante $p = [v_{10}, v_9, v_8, v_7, v_{b'}, v_1]$ na Figura C.7(b). Recuperar o caminho aumentante no grafo original a partir de p pode ser feito através de repetidas aplicações da construção existente na prova do teorema C.2.1 que establece a existência deste caminho. Para mostrar o princípio envolvido neste processo irá ser usado o exemplo das Figuras C.6 e C.7.

No caminho $p = [v_{10}, v_9, v_8, v_7, v_{b'}, v_1]$ existe um vértice flor $v_{b'}$. Primeiro encontra-se o vértice que sucede a $v_{b'}$ no caminho e é conectado com $v_{b'}$ através de uma aresta livre - v_1 no exemplo. De seguida, encontra-se o vértice em b' que é adjacente (no grafo antes do encolhimento de b') a v_1. Isto nem sempre será simples porque, em vez de se ter um vértice

v_1, pode-se ter um outro vértice flor. Uma vez encontrado o vértice de b' adjacente a v_1 (neste caso v_2 ou v_3); escolhe-se um deles, por exemplo v_2, e encontra-se o único caminho da base de b' até v_2 cuja aresta final é uma aresta emparelhada, $[v_b, v_3, v_2]$ no exemplo em questão. Sendo assim, substitui-se $v_{b'}$ em p por $[v_b, v_3, v_2]$ e obtém-se $p' = [v_{10}, v_9, v_8, v_7, v_b, v_3, v_2, v_1]$.

Agora é necessário repetir o mesmo processo para substituir v_b no caminho p'. Neste momento v_3 é o vértice que está conectado com v_b através de uma aresta livre. Verifica-se que o vértice de b que é adjacente a v_3 é v_4. Então encontra-se o único caminho dentro de b que vai desde a base v_6 até ao vértice v_4 e termina com uma aresta emparelhada, nomeadamente, $[v_6, v_5, v_4]$. Finalmente, substituí-se v_b em p' por este caminho e obtém-se o caminho aumentante final $p'' = [v_{10}, v_9, v_8, v_7, v_6, v_5, v_4, v_3, v_2, v_1]$ através do qual o procedimento AUGMENT (referido no algoritmo de emparelhamento para grafos bipartidos) aumenta o emparelhamento atual.

O Algoritmo completo é apresentado de seguida:

Algorithm 3 Algoritmo Edmonds Grafos Gerais

Input: Um grafo Bipartido $G = (V, E)$
Output: Emparelhamento máximo de G; representado na matriz *mate*

1: **Procedimento** $EMPARELHAMENTO\_GERAL$ $(G = (V, E))$:
2: **for** $v \in V \cup U$ do $mate[v] := 0$, $considerado[v] := 0$;
3: etapa: **while** existir um $u \in V$ com $considerado[u] = 0$ e $mate[u] = 0$ **do**
4: $considerado[u] := 1$, A:= $\oslash$;
5: **for all** $v \in V$ **do** $exposto[v] := 0$; (iniciar a construção do digrafo auxiliar $G = (V, A)$)
6: **for all** $[v, w] \in E$ **do** (repetir para ambos $[v, w]$ $[w, v]$)
7: **if** $mate[w] = 0$ and $w \neq u$ **then** $exposto[v] := w$ **else**
8: textbfif $mate[w] \neq v, 0$ **then** $A := A \cup \{(v, mate[w])\}$;
9: **for all** $v \in V$ **do** $visto[v] = 0$;
10: $Q := \{u\}$; $etiqueta[u] := 0$; **if** $exposto[u] \neq 0$ **then** AUGMENT(u), ir para etapa; (iniciar a pesquisa)
11: **while** $Q \neq \oslash$ **do**
12: seja v um vértice de Q; (Q é uma fila)
13: remover v de Q;
14: **for all** vértices não etiquetados $w \in V$ tais que $(v, w) \in A$ **do**
15: $Q := Q \cup \{w\}$, $etiqueta[w] := v$;
16: $visto[mate[w]] := 1$;
17: **if** $exposto[w] \neq 0$ **then** AUGMENT(w), ir para etapa;
18: **if** $visto[w] = 1$ **then** blossom(w)

Pelo **Teorema 10.6** descrito em (Papadimitriou and Kenneth, 1998), este algoritmo encontra corretamente o emparelhamento máximo num grafo $G(V, E)$ em tempo $O(|V|^4)$.

Exemplo de Aplicação do Algoritmo de Edmonds para grafos gerais

Para que se torne um pouco mais claro (dada a complexidade do algoritmo), apresenta-se um exemplo de aplicação do algoritmo descrito.
Considere-se o grafo da Figura C.13.

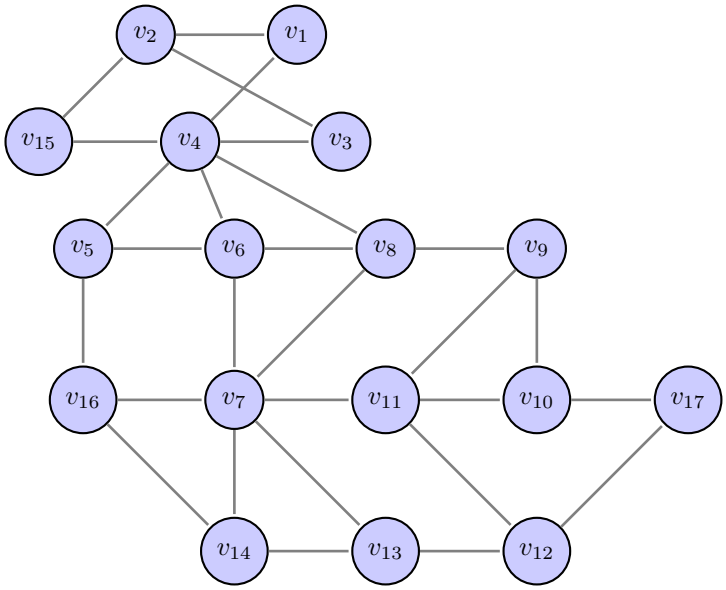


Figura C.13: Exemplo de Aplicação Algoritmo de Edmonds para grafos gerais

As primeiras sete etapas são triviais, uma vez que os vértices estão todos expostos. Os caminhos aumentantes consistem apenas em arestas individuais e são encontradas de forma imediata. O resultado destas sete etapas pode ser visualizado na Figura C.14.

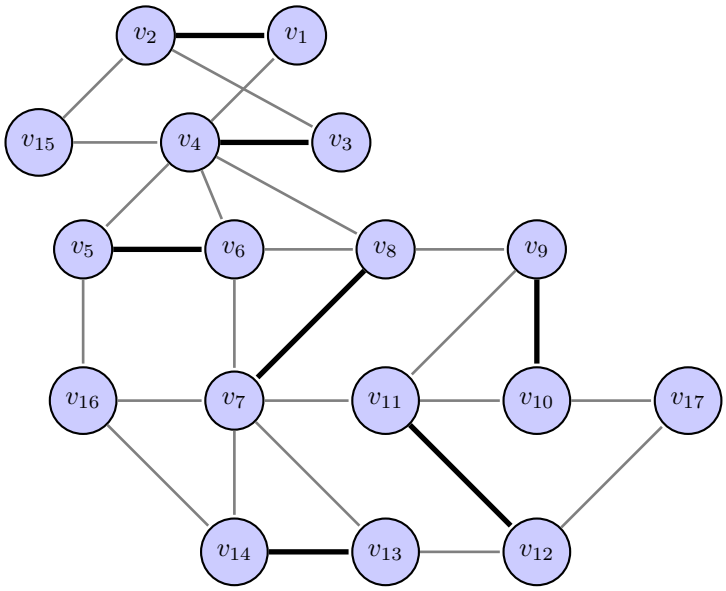


Figura C.14: As primeiras 7 etapas do Algoritmo de Edmonds para grafos gerais

Na próxima etapa pega-se num vértice exposto e que ainda não tenha sido considerado. Os vértices que estão nestas condições são v_{15}, v_{16} e v_{17}. Considera-se v_{15} e constrói-se o digrafo auxiliar relativamente ao emparelhamento atual, passos (5) até (9) (Figura C.15). Os vértices com $exposto[v] \neq 0$ são v_5, v_7, v_{10}, v_{12} e v_{14} (vértices alvo).

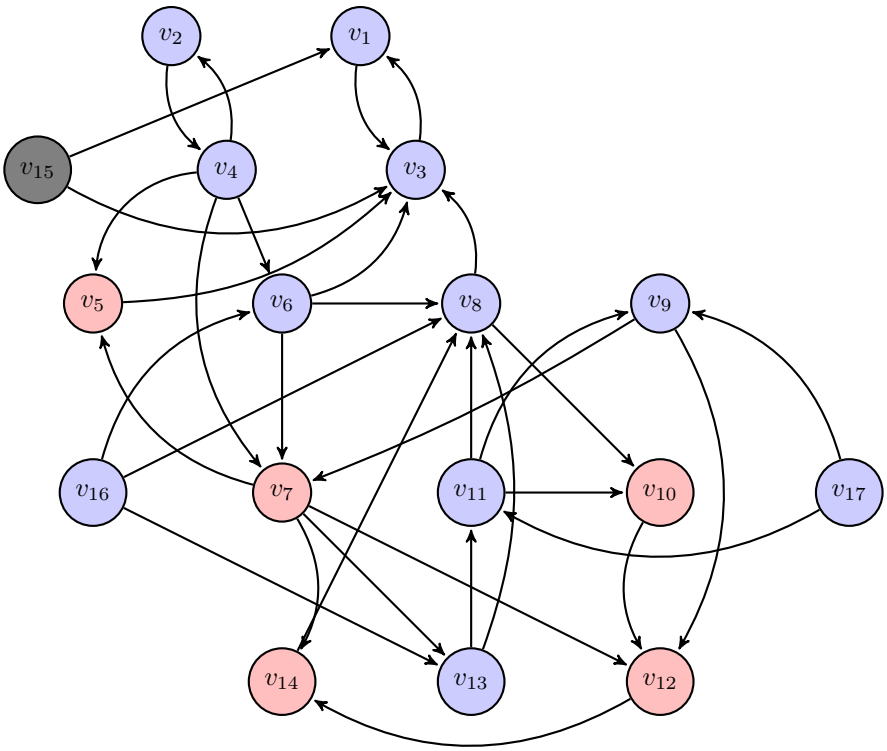
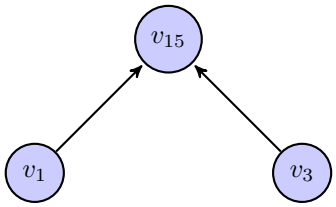


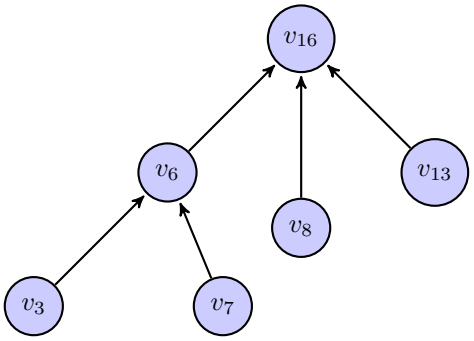
Figura C.15: Digrafo Auxiliar Exemplo C.13

Mostra-se de seguida a pesquisa a partir de v_{15} (um arco de v para u significa $etiqueta[u] = v$; isto é o inverso de um arco no digrafo auxiliar)



Neste momento a pesquisa termina sem sucesso, porque não se encontrou nenhum caminho aumentante (nenhum caminho de v_{15} até um vértice alvo), nenhuma flor e $Q = \oslash$. Portanto v_{15} é abandonado; e não volta a ser considerado como vértice inicial para uma pesquisa.

O seguinte vértice exposto ainda não considerado é v_{16}. Começa-se então a pesquisar no digrafo auxiliar a partir de v_{16}, com vértices alvo v_2, v_4, v_{10}, v_{12}. Quando se remove v_{16} de Q, pelos passos (14) até (16) os vértices v_6, v_8 e v_{13} são inseridos em Q e $visto[v_5] = visto[v_7] = visto[v_{14}] = 1$, porque são os "pares"dos vértices inseridos em Q, passos (14) até (16). De seguida, remove-se v_6 de Q. Isto faz com que v_3 seja inserido e que $visto[v_4] = 1$. Depois analisa-se v_7. Esta situação está representada na seguinte figura:



$$Q = \{v_8, v_{13}, v_3, v_7\}$$

Neste momento nota-se que $visto[v_7] = 1$; então v_7 é o "par"de um vértice exposto - v_8- e foi descoberta uma flor, passo (18). Para encontrar todos os vértices da flor segue-se a matriz *etiqueta* "para trás"(isto é backtracking na matriz *etiqueta*) a partir dos vértices v_7 e v_8 ("par"de v_7). O primeiro vértice comum nestes "caminhos para trás"é v_{16}, então v_{16} é a base da flor. Todos os vértices encontrados neste "backtracking"e os seus "pares"constituem a flor. Esta flor é denotada por um novo vértice v_{18}. Substituem-se em todas as entradas de Q e da matriz *etiqueta* os vértices pertencentes à flor por v_{18} e retoma-se a pesquisa. O grafo auxiliar modificado é ilustrado na Figura C.16.

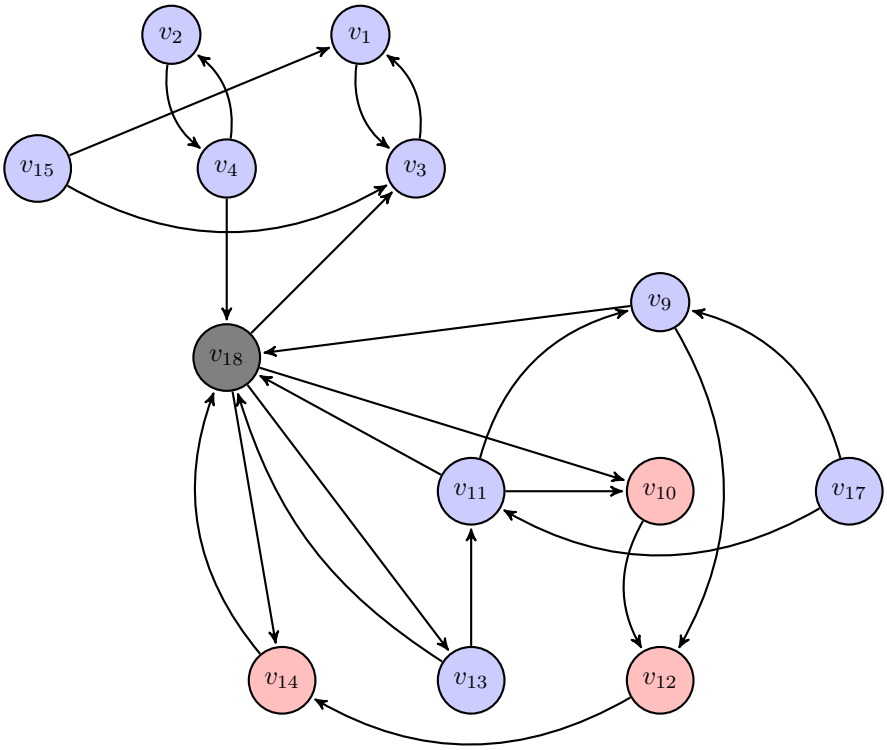
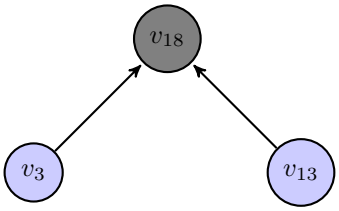


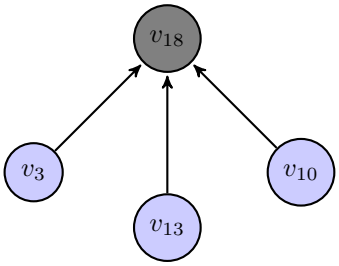
Figura C.16: Digrafo Auxiliar Modificado Exemplo C.13

A pesquisa continua a partir do seguinte estado:



$$Q = \{v_{18}, v_{13}, v_3\}$$

De seguida, remove-se v_{18} da fila, e adiciona-se v_{10}. O vértice v_{10} possui um vértice exposto, isto é, $exposto[v_{10}] = v_{17} \neq 0$.



Sendo assim, encontrou-se um caminho aumentante no grafo atual, nomeadamente, $p' = [v_{18}, mate[v_{10}], v_{10}, exposto[v_{10}]] = [v_{18}, v_9, v_{10}, v_{17}]$, passo (17). Para substituir p' no grafo original, analisam-se todas as arestas incidentes em v_9 e descobre-se que $[v_{18}, v_9]$ de facto corresponde à aresta original $[v_8, v_9]$; além disso, o caminho a partir da base de v_{18} até v_8 que termina com uma aresta emparelhada é $[v_{16}, v_7, v_8]$. Então substitui-se v_{18} em p' por $[v_{16}, v_7, v_8]$ e obtém-se o caminho aumentante final $p = [v_{16}, v_7, v_8, v_9, v_{10}, v_{17}]$. Por último, aumenta-se o emparelhamento atual através de p (Figura C.17) e descobre-se que não existe nenhum vértice não emparelhado e não considerado em G, por isso o emparelhamento atual é ótimo.

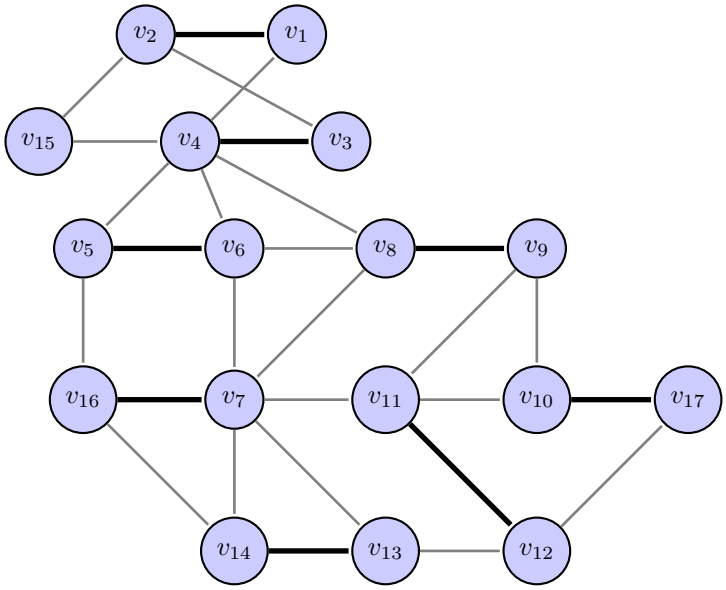


Figura C.17: Emparelhamento Ótimo Exemplo C.13

Anexo D

Resultados referentes às instâncias de tamanho 10

Neste apêndice apresenta-se a informação referente à aplicação da Formulação Ciclo e do algoritmo desenvolvido às instâncias de tamanho 10. Para cada instância é apresentado o número de ciclos de tamanho 3 que esta possui. No caso da Formulação Ciclo a única informação obtida é o tempo de execução, enquanto que, para cada uma das versões do algoritmo desenvolvido (Pesquisa em Largura e Pesquisa em Profundidade), sabe-se o número de nós visitados e o tempo dispendido para se encontrar a melhor solução. Por último é apresentada uma tabela onde para cada instância está representada a média do número de ciclos com vértices comuns.

Instância Gerada	**Número de Ciclos de tamanho 3**	**Pesquisa em Profundidade**	
		nós visitados	**tempo execução**
1	0	1	0.00388
2	0	1	0.04703
3	0	1	0.05555
4	0	1	0.03231
5	0	1	0.04672
6	4	9	0.03188
7	0	1	0.04459
8	0	1	0.05589
9	4	2	0.04731
10	0	1	0.05188
11	4	2	0.06898
12	0	1	0.04646
13	0	1	0.06250
14	0	1	0.06149
15	0	1	0.05208
16	4	6	0.05407
17	2	2	0.05023
18	0	1	0.08259
19	5	1	0.06925

Instância Gerada	Número de Ciclos de tamanho 3	Pesquisa em Profundidade	
		nós visitados	tempo execução
20	0	1	0.05881
21	0	1	0.02548
22	11	2	0.10878
23	2	2	0.04943
24	5	2	0.06456
25	10	2	0.07936
26	0	1	0.04004
27	2	5	0.05835
28	18	3	0.04783
29	0	1	0.05869
30	0	1	0.04797
31	0	1	0.03763
32	0	1	0.0482
33	0	1	0.02561
34	10	6	0.06108
35	0	1	0.03851
36	8	2	0.02937
37	2	2	0.04962
38	0	1	0.06092
39	0	1	0.02321
40	3	1	0.04694
41	3	49	0.10814
42	0	1	0.02734
43	6	3	0.01226
44	11	1	0.07057
45	0	1	0.02729
46	13	3	0.07006
47	0	1	0.04049
48	4	1	0.08431
49	0	1	0.06066
50	3	2	0.05585

Instância Gerada	Pesquisa em Largura		Formulação Ciclo
	nós visitados	tempo execução	
1	1	0.00776	0.01798
2	1	0.00705	0.07455
3	1	0.00629	0.02395
4	1	0.00334	0.05637
5	1	0.00467	0.06128
6	9	0.01539	0.03706
7	1	0.00526	0.04317

Instância Gerada	**Pesquisa em Largura**		**Formulação Ciclo**
	nós visitados	**tempo execução**	
8	1	0.00582	0.03982
9	3	0.0102	0.04011
10	1	0.00626	0.01817
11	3	0.01105	0.03409
12	1	0.00613	0.01458
13	1	0.00655	0.01607
14	1	0.00313	0.06696
15	1	0.00652	0.03580
16	7	0.02025	0.03163
17	3	0.01012	0.04429
18	1	0.00657	0.04229
19	1	0.01398	0.04256
20	1	0.00525	0.03561
21	1	0.00549	0.03547
22	3	0.01413	0.06798
23	3	0.00987	0.04395
24	3	0.01319	0.05443
25	3	0.01403	0.04042
26	1	0.00452	0.03255
27	5	0.01516	0.04449
28	7	0.01863	0.04232
29	1	0.00662	0.04232
30	1	0.00548	0.0405
31	1	0.00646	0.03712
32	1	0.00486	0.04091
33	1	0.00532	0.03205
34	5	0.02186	0.0435
35	1	0.00568	0.03482
36	3	0.01281	0.04139
37	3	0.01078	0.04395
38	1	0.00638	0.03429
39	1	0.00667	0.04476
40	1	0.00729	0.03636
41	37	0.06926	0.04245
42	1	0.00578	0.03525
43	7	0.01915	0.03602
44	1	0.0088	0.03895
45	1	0.00506	0.03385
46	7	0.01831	0.04483
47	1	0.00493	0.0352
48	1	0.00812	0.04564
49	1	0.00526	0.04159
50	3	0.00974	0.02271

Tabela D.1: Resultados referentes ao tempo de execução das instâncias de tamanho 10

Instância Gerada	**Média de ciclos com vértices comuns**	**Instância Gerada**	**Média de ciclos com vértices comuns**
1	0	26	0
2	0	27	1
3	0	28	16
4	0	29	0
5	0	30	0
6	3	31	0
7	0	32	0
8	0	33	0
9	3	34	8
10	0	35	0
11	3	36	7
12	0	37	1
13	0	38	0
14	0	39	0
15	0	40	2
16	0	41	12
17	1	42	0
18	0	43	4
19	4	44	10
20	0	45	0
21	0	46	11
22	10	47	0
23	1	48	3
24	4	49	0
25	9	50	2

Tabela D.2: Resultados referentes aos ciclos com vértices comuns das instâncias de tamanho 10

Anexo E

Resultados referentes às instâncias de tamanho 20

Neste apêndice apresenta-se a informação referente à aplicação da Formulação Ciclo e do algoritmo desenvolvido às instâncias de tamanho 20. Para cada instância é apresentado o número de ciclos de tamanho 3 que esta possui. No caso das duas versões do algoritmo desenvolvido (Pesquisa em Largura e Pesquisa em Profundidade) sabe-se o número de nós visitados e o tempo dispendido para se encontrar a melhor solução. Enquanto que, para a Formulação Ciclo a única informação obtida é o tempo de execução. Por último é apresentada uma tabela onde para cada instância está representada a média do número de ciclos com vértices comuns.

Instância Gerada	**Número de Ciclos de tamanho 3**	**Pesquisa em Profundidade**	
		nós visitados	**tempo execução**
1	35	1	0.0419
2	3	2	0.06281
3	55	62	0.2711
4	4	9	0.06201
5	4	2	0.04306
6	35	8	0.0925
7	20	1	0.05865
8	44	5	0.07902
9	49	343	1.16913
10	19	4	0.1214
11	0	1	0.02609
12	8	17	0.14919
13	20	43	0.1596
14	40	193	0.75267
15	145	43719	101.43108
16	20	41	0.16692
17	6	13	0.06567
18	21	13	0.11808
19	4	1	0.0597

Instância Gerada	**Número de Ciclos de tamanho 3**	**Pesquisa em Profundidade**	
		nós visitados	**tempo execução**
20	31	7	0.06404
21	69	1601	4.25296
22	115	86	0.39156
23	5	1	0.04598
24	2	1	0.03522
25	2	2	0.09167
26	105	17	0.12502
27	4	1	0.05169
28	28	89	0.30246
29	78	248	0.79513
30	62	1	0.082
31	13	1	0.07719
32	47	191	0.6393
33	1	2	0.04409
34	9	19	0.1105
35	85	1	0.11214
36	6	1	0.05945
37	71	4	0.07764
38	5	2	0.05938
39	0	1	0.05094
40	52	383	1.13701
41	20	2	0.07138
42	17	8	0.07686
43	40	1	0.07667
44	6	1	0.05393
45	64	4	0.12127
46	12	25	0.12847
47	60	855	3.20757
48	6	2	0.03802
49	0	1	0.05512
50	70	880	2.42548

Instância Gerada	**Pesquisa em Largura**		**Formulação Ciclo**
	nós visitados	**tempo execução**	
1	1	0.02659	0.16265
2	3	0.02921	0.04698
3	119	0.39347	0.05565
4	9	0.0337	0.04633
5	3	0.02464	0.04262
6	23	0.11466	0.06129
7	1	0.02228	0.05186

Instância Gerada	**Pesquisa em Largura**		**Formulação Ciclo**
	nós visitados	**tempo execução**	
8	9	0.05266	0.05513
9	343	1.09594	0.05022
10	5	0.03893	0.04735
11	1	0.01629	0.04437
12	17	0.06932	0.04797
13	43	0.1508	0.04287
14	175	0.63789	0.08587
15	43719	93.46014	0.06844
16	41	0.14754	0.04505
17	13	0.05703	0.02022
18	31	0.15959	0.04838
19	1	0.02247	0.02049
20	19	0.09303	0.04912
21	1625	4.22027	0.06021
22	79	0.43423	0.12829
23	1	0.01394	0.01542
24	1	0.02052	0.01327
25	3	0.03976	0.04803
26	55	0.31652	0.06237
27	1	0.01279	0.03999
28	89	0.22995	0.05287
29	137	0.54405	0.0591
30	1	0.03549	0.05105
31	1	0.02799	0.05364
32	175	0.60084	0.05517
33	3	0.01628	0.0288
34	19	0.06759	0.04094
35	1	0.04547	0.03224
36	1	0.02216	0.07959
37	15	0.10378	0.06595
38	3	0.02359	0.02104
39	1	0.0297	0.04494
40	363	1.0922	0.07403
41	3	0.04383	0.08896
42	9	0.04452	0.05754
43	1	0.03358	0.05482
44	1	0.01481	0.0488
45	15	0.15421	0.04546
46	25	0.08613	0.02775
47	855	3.36921	0.04243
48	3	0.03023	0.04524
49	1	0.01638	0.04256
50	433	1.48489	0.03746

Tabela E.1: Resultados referentes ao tempo de execução das instâncias de tamanho 20

Instância Gerada	**Média de ciclos com vértices comuns**	**Instância Gerada**	**Média de ciclos com vértices comuns**
1	30	26	72
2	2	27	3
3	40	28	25
4	3	29	53
5	0	30	48
6	27	31	7
7	19	32	38
8	41	33	0
9	38	34	8
10	18	35	64
11	0	36	15
12	7	37	53
13	16	38	4
14	30	39	0
15	96	40	42
16	17	41	12
17	5	42	16
18	14	43	30
19	3	44	5
20	26	45	49
21	49	46	11
22	78	47	36
23	4	48	5
24	1	49	0
25	1	50	39

Tabela E.2: Resultados referentes aos ciclos com vértices comuns das instâncias de tamanho 20

Anexo F

Resultados referentes às instâncias de tamanho 30

Neste apêndice apresenta-se a informação referente à aplicação da Formulação Ciclo e do algoritmo desenvolvido às instâncias de tamanho 30. Para cada instância é apresentado o número de ciclos de tamanho 3 que esta possui. No caso da Formulação Ciclo a única informação obtida é o tempo de execução. Enquanto que, para cada uma das versões do algoritmo desenvolvido (Pesquisa em Largura e Pesquisa em Profundidade) sabe-se o número de nós visitados e o tempo dispendido para se encontrar a melhor solução. Por último é apresentada uma tabela onde para cada instância está representada a média do número de ciclos com vértices comuns.

Instância Gerada	**Número de Ciclos de tamanho 3**	**Pesquisa em Profundidade**	
		nós visitados	**tempo execução**
1	12	557	5.62974
2	7	15	0.23053
3	16	1	0.09017
4	43	43	0.30933
5	32	67	0.32996
6	15	1	0.09955
7	2	5	0.1334
8	53	14	0.18927
9	174	18753	92.07173
10	43	87	0.63627
11	12	25	0.21168
12	50	213	1.29093
13	25	329	1.569
14	51	1	0.11659
15	78	3	0.12545
16	217	120	0.84515
17	0	1	0.0721
18	106	14	0.27897
19	108	1283	8.27474

Instância Gerada	**Número de Ciclos de tamanho 3**	**Pesquisa em Profundidade**	
		nós visitados	**tempo execução**
20	39	3	0.14573
21	5	19	0.19586
22	69	215	1.35664
23	106	3423	21.17575
24	236	1790	8.28072
25	164	8	0.1955
26	20	2	0.09034
27	296	280	0.81652
28	27	69	0.70463
29	16	3	0.21302
30	50	1	0.1426
31	87	3	0.18756
32	54	42	0.33956
33	124	7993	36.95197
34	72	9	0.19796
35	67	1	0.15128
36	45	273	1.61214
37	111	401	2.61732
38	81	97	0.92711
39	141	24	0.49504
40	44	1	0.11795
41	184	56	0.46073
42	306	111	1.00997
43	105	7485	48.37776
44	142	7	0.15563
45	24	237	1.22964
46	177	1	0.16874
47	84	2	0.14367
48	91	1	0.11149
49	58	1185	5.78268
50	24	151	1.31315

Instância Gerada	**Pesquisa em Largura**		**Formulação Ciclo**
	nós visitados	**tempo execução**	
1	1443	11.87998	0.05479
2	15	0.21097	0.04453
3	1	0.07977	0.04655
4	97	0.72724	0.05285
5	69	0.29754	0.04532
6	1	0.09082	0.05287
7	5	0.12953	0.04996

Instância Gerada	**Pesquisa em Largura**		**Formulação Ciclo**
	nós visitados	**tempo execução**	
8	55	0.53629	0.06649
9	18715	92.59169	0.0881
10	87	0.62361	0.05816
11	25	0.19833	0.05318
12	213	1.2623	0.06067
13	331	1.50475	0.02825
14	1	0.13689	0.05728
15	7	0.19989	0.03279
16	107	0.98421	0.07855
17	1	0.0666	0.02523
18	43	0.58858	0.08001
19	1287	8.2844	0.0385
20	7	0.17577	0.05897
21	19	0.16607	0.04976
22	197	1.21054	0.10642
23	2719	16.50533	0.08138
24	16289	76.88689	0.10573
25	21	0.34452	0.06881
26	3	0.09741	0.06002
27	29	0.49799	0.14278
28	65	0.68006	0.05801
29	7	0.26452	0.05678
30	1	0.14205	0.06325
31	7	0.24989	0.06918
32	241	1.67193	0.06583
33	7983	33.06385	0.07767
34	17	0.26584	0.07018
35	1	0.16357	0.0665
36	303	1.68492	0.0605
37	273	2.48382	0.0814
38	151	1.38139	0.07538
39	25	0.54668	0.08386
40	1	0.1131	0.06488
41	127	1.16338	0.0678
42	75	1.14998	0.12989
43	7497	46.93683	0.06259
44	37	0.34029	0.06807
45	207	1.06755	0.02791
46	1	0.17056	0.09268
47	3	0.15567	0.10239
48	1	0.10987	0.06828
49	1189	5.50007	0.038
50	151	1.28915	0.03547

Tabela F.1: Resultados referentes ao tempo de execução das instâncias de tamanho 30

Instância Gerada	**Média de ciclos com vértices comuns**	**Instância Gerada**	**Média de ciclos com vértices comuns**
1	77	26	19
2	6	27	149
3	12	28	21
4	32	29	11
5	30	30	49
6	14	31	46
7	1	32	31
8	37	33	73
9	133	34	57
10	42	35	44
11	8	36	33
12	46	37	73
13	14	38	62
14	35	39	90
15	61	40	29
16	125	41	112
17	0	42	138
18	73	43	74
19	73	44	100
20	30	45	15
21	2	46	132
22	56	47	59
23	68	48	58
24	135	49	41
25	106	50	16

Tabela F.2: Resultados referentes aos ciclos com vértices comuns das instâncias de tamanho 30

Anexo G

Resultados referentes às instâncias de tamanho 40

Neste apêndice apresenta-se a informação referente à aplicação da Formulação Ciclo e do algoritmo desenvolvido às instâncias de tamanho 40. Para cada instância é apresentado o número de ciclos de tamanho 3 que esta possui. No caso das duas versões do algoritmo desenvolvido (Pesquisa em Largura e Pesquisa em Profundidade) sabe-se o número de nós visitados e o tempo dispendido para se encontrar a melhor solução. Enquanto que, para a Formulação Ciclo a única informação obtida é o tempo de execução. Por último é apresentada uma tabela onde para cada instância está representada a média do número de ciclos com vértices comuns.

Instância Gerada	**Número de Ciclos de tamanho 3**	**Pesquisa em Profundidade**	
		nós visitados	**tempo execução**
1	69	447	6.63508
2	11	2	0.16643
3	209	3414	22.05271
4	339	175253	1720.00717
5	336	144	2.50532
6	64	45	0.76443
7	143	5327	51.71752
8	57	1197	12.50857
9	86	2093	25.58371
10	212	92413	928.75586
11	198	1335	13.67047
12	114	63395	2416.81327
13	267	37	0.77354
14	186	148	1.25716
15	365	701	4.40288
16	296	258639	1801.83266
17	112	6109	69.27653
18	320	225	2.66216
19	48	225	2.66216

Instância Gerada	**Número de Ciclos de tamanho 3**	**Pesquisa em Profundidade**	
		nós visitados	**tempo execução**
20	292	107	1.79242
21	33	145	1.33869
22	308	1	0.45355
23	20	73	1.13894
24	152	7827	71.12043
25	69	1	0.22216
26	71	3	0.39533
27	112	1	0.24403
28	264	123593	1231.37789
29	165	11645	114.27011
30	103	4553	48.00469

Instância Gerada	**Pesquisa em Largura**		**Formulação Ciclo**
	nós visitados	**tempo execução**	
1	447	6.47679	0.06313
2	3	0.20325	0.08737
3	525	6.38553	0.12522
4	174723	1778.38437	0.16548
5	1159	15.67705	0.15338
6	129	1.95182	0.07121
7	4379	44.12302	0.07739
8	1221	12.33787	0.07124
9	2023	24.8856	0.06032
10	104471	1037.99592	0.10729
11	5465	59.16812	0.10366
12	67155	2160.09513	0.09227
13	173	3.11739	0.16394
14	825	7.42559	0.13118
15	249	5.86333	0.16472
16	239699	1841.4675	0.12156
17	6841	79.10236	0.08769
18	489	6.77717	0.14586
19	233	2.63964	0.06354
20	263	6.07896	0.13831
21	145	1.35818	0.08364
22	1	0.45692	0.14568

Instância Gerada	**Pesquisa em Largura**		**Formulação Ciclo**
	nós visitados	**tempo execução**	
23	57	0.93694	0.0613
24	997	11.62047	0.09732
25	1	0.22923	0.09502
26	7	0.44753	0.07837
27	1	0.24035	0.07755
28	32473	400.68541	0.12285
29	11687	115.68923	0.12666
30	2485	31.27545	0.08009

Tabela G.1: Resultados referentes ao tempo de execução das instâncias de tamanho 40

Instância Gerada	**Média de ciclos com vértices comuns**	**Instância Gerada**	**Média de ciclos com vértices comuns**
1	60	16	159
2	7	17	60
3	107	18	178
4	181	19	41
5	161	20	133
6	47	21	27
7	81	22	153
8	36	23	11
9	64	24	81
10	117	25	59
11	110	26	56
12	48	27	86
13	158	28	140
14	126	29	98
15	172	30	54

Tabela G.2: Resultados referentes aos ciclos com vértices comuns das instâncias de tamanho 40